Zurich University
of Applied Sciences

School of Engineering

IAMP Institute of Applied Mathematics and Physics

Bachelor Thesis B.Sc. Computer Science

Simultaneous Localization and Mapping Algorithm for an Autonomous Race Car

Authors	Andreas Sprecher Andrea Stöckli
Main Supervisor	Dr. Monika Ulrike Reif
Industrial partner	Zurich UAS Racing
External supervisor	Luis Miguel Da Silva Miranda
Date	17.07.2023

Zurich University
of Applied Sciences

School of Engineering
IAMP Institute of Applied Mathematics and Physics

DECLARATION OF ORIGINALITY

Bachelor's Thesis at the School of Engineering

By submitting this bachelor thesis, the undersigned student confirm that this work is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:	**Signature:**
	Andreas Sprecher
....	....
	Andrea Stöckli
....	....

Abstract

The field of autonomous robotics, an emergent discipline within engineering, has attracted substantial interest in recent years. One of the notable platforms that challenge and stimulate progress in this area is the Formula Student driverless competition. This international contest prompts students to engineer, design, and fabricate a formula-style, single-seater race car capable of autonomous driving. The core of these driverless systems relies heavily on Simultaneous Localization and Mapping algorithms (SLAM), making them an essential focus of research and development for all types of autonomous robots. Literature shows a variety of approaches to tackle the SLAM problem. In light of this diversity, we selected the most promising algorithms suitable for our specific use case, considering numerous requirements and restrictions related to the entire driverless platform developed by our team at Zurich UAS Racing. Besides the implementation of EKF SLAM and FastSLAM, specific metrics and test cases were established that allow for their verification and comparative analysis. We further developed supporting code, such as a car simulator and an algorithm evaluator, as well as a software pipeline to automate the build and release processes of the whole driverless system. Our implementation demonstrated that the chosen SLAM algorithms produced satisfactory results by generating maps suitable for navigation. Notably, our experimental results showed FastSLAM outperforming EKF SLAM, thus it was selected as the algorithm of choice for the upcoming driverless competition. The successful implementation highlights how these algorithms, particularly FastSLAM, can play a key role in elevating the performance and reliability of the autonomous driving system at Zurich UAS Racing.

Keywords: Autonomous Robots; SLAM; Formula Student

Contents

Chapter 1

Introduction

1.1 Motivation

Autonomous driving is a rapidly evolving technology that has the potential to revolutionize the way we move and interact with our environment. Self-driving vehicles offer numerous benefits, including increased safety, reduced traffic congestion and improved mobility for those who cannot or should not drive [69], [35]. With the development of advanced sensors, computer vision and machine learning algorithms, autonomous vehicles can perceive and interpret their surroundings, make decisions and navigate complex environments without human intervention [49], [12]. Companies in the field, such as Waymo, Cruise, Mercedes and BMW, are developing autonomous driving systems that utilize a combination of sensor technologies and advanced algorithms. Despite the tremendous progress made in autonomous driving technology, significant challenges remain in achieving fully autonomous driving capabilities. Some of these challenges include ensuring the safety and reliability of self-driving vehicles, addressing ethical and legal issues and establishing appropriate regulations and standards [69]. Competitions accelerate technological advances in various areas. For example, technologies first introduced in Formula One, such as carbon fiber composite materials and kinetic energy recovery systems, were later used in commercial vehicles [52], [37]. A competitive environment enables clear parameters for success and offers the possibility for scientific studies. Therefore, competitions in which autonomous racing cars compete against each other add valuable contributions to current research in the field of autonomous driving [6].

1.2 Zurich UAS Racing

A highly regarded challenge is the international Formula Student (FS) engineering competition where teams of students design, build and race a formula-style race car. Besides the Internal Combustion Engine Vehicles (CV) and Electric Vehicles (EV) events with human drivers, the Driverless Cup (DC) is a competition that challenges student teams to create an autonomous system that manoeuvres the race car. The event provides a platform for students to improve and showcase their innovative ideas and technical skills in the field of autonomous driving.

Launched in 2019, the Zurich University of Applied Sciences Racing (ZUR) Team from Winterthur built a fully functional car in 2021 seen in Figure 1.1a and successfully participated at the EV events in 2022 with the car illustrated in Figure 1.1b. The goal of ZUR is to improve the performance of 2022 at the EV events in 2023 and achieve solid results at the DC as well with a new and improved car. As part of the ZUR, the authors of this thesis want to contribute to the current development in the field of autonomous driving and help ZUR to achieve their goals.

(a) ZUR Car 2021

(b) ZUR Car 2022

Figure 1.1: Past ZUR Cars (a) 2021 (b) 2022

1.2.1 Driverless Team

The development of autonomous racing vehicles is an exciting and challenging endeavour. The driverless team is organized in groups that develop object perception, Simultaneous Localization and Mapping (SLAM), trajectory planning and car controls. Object detection identifies the relative position and colour of the cones that indicate the race track. With that information, a SLAM algorithm enables the vehicle to create a map of its surroundings while simultaneously determining its position within that map. Trajectory planning involves generating optimal paths in the map for the vehicle to follow. Car controls involve the implementation of algorithms that manage the vehicle's throttle, braking, and steering systems to follow the path determined by the trajectory planning algorithm.

1.2.2 SLAM Group

The SLAM group at ZUR focuses on developing a robust SLAM algorithm for autonomous racing. Two projects have been conducted by the group so far. These projects are stepping stones towards developing a fully functional SLAM algorithm that the team aims to have operational by the summer races. This bachelor thesis, which is part of a collaborative effort with the ZHAW Institute of Applied Mathematics and Physics (IAMP), aims to extend the existing body of work and contribute to the SLAM development at ZUR. Achieving a reliable SLAM algorithm, capable of creating accurate maps and determining the car's position within these environments, is a pivotal part of the driverless team. The results of this work will be critical for future developments and adaptations in autonomous driving for the ZUR team.

1.3 Related Work

SLAM has its roots in the late 1980s, with early research focused on probabilistic robot representation and mapping [56], [17], [55]. The term SLAM was introduced in 1995, and early approaches relied on Kalman Filters [18]. Another SLAM approach uses smoothing instead of filtering [33]. Modern advancements in computational capabilities have made smoothing-based approaches like GraphSLAM popular [22].

The optimal SLAM approach depends on context, including factors like available sensors and environment size. Due to its racing application and constraints, FS Driverless (DV) vehicles have specific requirements. Existing work in the FS DV context has explored FastSLAM [28], ORB-SLAM2 [3], EKF SLAM [65], [9], and comparisons between algorithms [31]. SLAM approaches have also been evaluated in other racing contexts like Roborace [45] and the DARPA challenge [15], but differ significantly from the FS application.

1.4 Project Goals and Scope

As part of the SLAM group in the ZUR team, the aim is to deliver an appropriate and fully functional SLAM algorithm that meets all requirements for the DV events. Choosing the most suitable SLAM algorithm is challenging, given the diverse set of available algorithms. This bachelor thesis aims to evaluate and implement multiple SLAM algorithms. This results in the foundation to confidently choose the best algorithm to use at the FS Spain 2023 DC for the ZUR team. With this goal in mind, the following research questions were defined:

1. What SLAM algorithms are in use for autonomous driving?
2. What are good metrics to evaluate SLAM algorithms for the DC?
3. Which self-implemented SLAM algorithm performs best on the defined metrics for relevant tracks?

Besides answering these research questions, the SLAM algorithms for ZUR will be integrated in the driverless software including control signal handling for the car, cone detection and trajectory planning.

The thesis starts with an overview of SLAM algorithms and their role, performance and problems in autonomous vehicle navigation. With the gathered information, the methodology to implement and evaluate SLAM algorithms is defined. In the implementation chapter, the SLAM algorithms implemented are documented. Next, the evaluation results are provided. In the discussion section, the results are classified and the best-fitting algorithm is presented afterwards in the conclusion chapter. Finally, possible future improvements are presented.

Overall, this bachelor thesis contributes a comprehensive evaluation of multiple SLAM algorithms for autonomous navigation in the context of FS racing. The findings will be of practical significance to the driverless team of ZUR.

Chapter 2

Background

2.1 Autonomous Systems

Autonomous systems are intelligent agents that operate with minimal human intervention. They are designed to perceive their environment, reason about it and make decisions to achieve specific goals. Agents interact with their environment through sensors and actuators. The agent's decision-making process is governed by its internal architecture, which could be based on logic, probability theory, or machine learning algorithms. The agent's performance is measured against a predefined performance measure, which allows for the evaluation and improvement of the system [51].

An essential aspect of autonomous systems is the ability to handle uncertainty in their environment. Probabilistic reasoning allows agents to model uncertain information and make decisions based on incomplete or noisy data. Probabilistic reasoning techniques, such as Bayesian networks and Markov models, are widely used in various applications, including robotics, computer vision and natural language processing [51].

Planning and learning are two key components of autonomous systems. Planning involves generating sequences of actions that allow the agent to achieve its goals, while learning enables the agent to improve its performance over time by adapting to new experiences. Examples of planning and learning algorithms are search algorithms, reinforcement learning and decision tree induction [51].

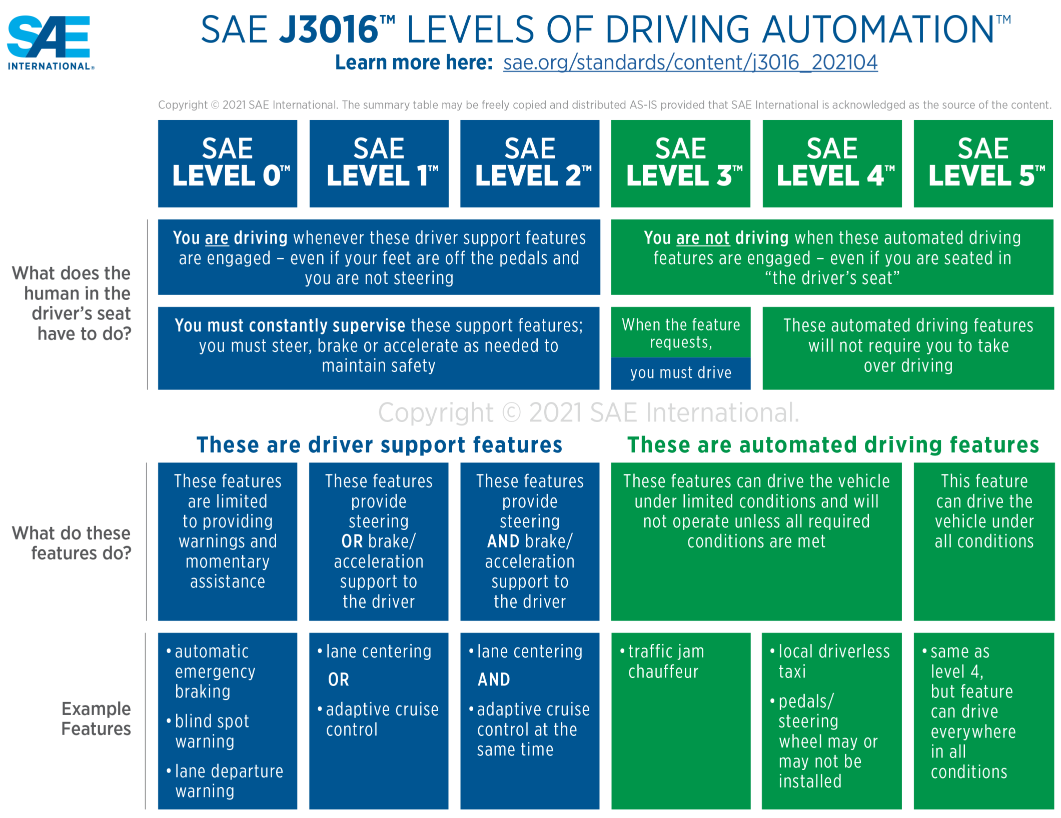
Autonomous vehicles represent a prominent application of autonomous systems, employing complex sensor suites and advanced algorithms to navigate safely and efficiently. Another example includes autonomous space robots, like Mars rovers, which navigate harsh terrains and conduct scientific experiments with minimal human intervention. Household helper robots and industrial automation robots also constitute significant applications of autonomous systems. They undertake various tasks such as vacuuming, lawn mowing or even assisting individuals with mobility challenges. Across these applications, a central challenge is the autonomous system's capacity to perceive its environment and accurately locate itself within it. This presents the SLAM problem as a pivotal hurdle to surmount. Solutions to the SLAM problem form basis for robust autonomous navigation, thereby placing these technologies at the forefront of autonomous system development [63].

2.1.1 Autonomous Cars

Autonomous cars, designed to navigate and operate without human intervention, utilize advanced sensors and Artificial Intelligence (AI) algorithms to perceive their environment, make decisions and control their movements. The Society of Automotive Engineers (SAE) has devised a classification system defining six levels of driving automation, ranging from no automation (Level 0) to full automation (Level 5) [25].

Levels of Driving Automation

The SAEs classification of driving automation, depicted in Figure 2.1, provides a comprehensive framework to understand the progression from traditional, manually-operated vehicles to fully autonomous cars.



SAE J3016™ LEVELS OF DRIVING AUTOMATION™

Learn more here: sae.org/standards/content/j3016_202104

	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			**These are automated driving features**		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/acceleration support to the driver	These features provide steering **AND** brake/acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 2.1: SAE J3016 Levels of Driving Automation [25]

Level 0 (No Automation) refers to vehicles lacking autonomous capabilities, while Level 1 (Driver Assistance) includes cars equipped with single-function systems, such as adaptive cruise control, that assist drivers with specific tasks. At Level 2 (Partial Automation), vehicles can control both steering and acceleration/deceleration, but driver engagement and monitoring are still required. With Level 3 (Conditional Automation), vehicles can perform all driving tasks under certain conditions, but drivers must be ready to take control when necessary. Advanced perception systems, including Light Detection and Ranging (LiDAR), radar, cameras and ultrasonic sensors, are vital at this level to provide a holistic understanding of the vehicle's surroundings. Level 4 (High Automation) vehicles can perform all driving tasks in specific circumstances without driver intervention. Finally, Level 5 (Full Automation) represents fully autonomous vehicles capable of performing all driving tasks in any situation without human intervention. This level calls for significant advancements in AI, computer vision and sensor technology to enable a complete understanding of the vehicle's environment and make appropriate, reliable and safe decisions [25].

General Approach

The design and implementation of autonomous vehicles involve a complex interplay of several subsystems responsible for perception, localization and mapping, path planning, decision-making and control tasks. An integral part of this architecture is the

SLAM system. SLAM refers to the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. The solution to this problem is a cornerstone for autonomous vehicles as it enables robust navigation and situational awareness. Figure 2.2 presents a general model of an autonomous car, encapsulating these key areas.

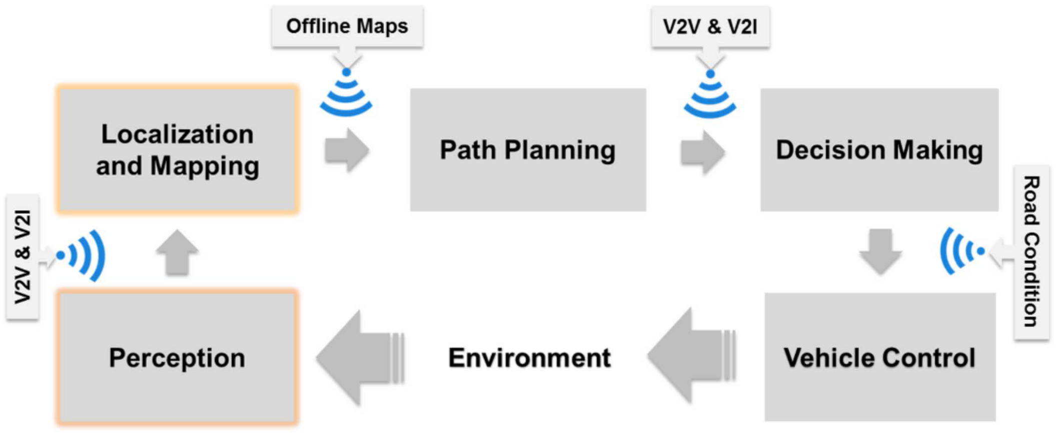


Figure 2.2: Overview of the autonomous navigation process [64]

The perception system acts as the autonomous vehicle's senses, interpreting raw sensor data from a range of sources, including LiDAR, cameras, radars and ultrasonic sensors. This data is processed and fused to form a comprehensive, dynamic model of the environment. Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) can help generate more sensor redundancies and increase traffic flow by creating a network of cars and infrastructure such as intersections and road or construction signs. The localization and mapping system identifies landmarks or features in the environment, tracking their positions over time to create a detailed map. Observing how these features move relative to the vehicle, it also reduces the vehicle's own movement and location. The process of path planning involves determining safe and efficient routes to the vehicle's intended destination using the map generated by the localization and mapping process, along with its knowledge of its current position. Decision-making involves determining the vehicle's actions based on its perceived environment and the driving mission. This process can include decisions on manoeuvres and interactions with other road users. Finally, the control system executes the decided actions, operating the vehicle's throttle, brakes and steering to ensure safe and efficient navigation. This system also maintains the stability and comfort of the vehicle during navigation [64].

Autonomous Cars in Use

The development of autonomous cars continues to evolve rapidly, driven by advancements in technology, changing societal needs and ongoing research into AI. These vehicles have the potential to revolutionize mobility, reducing the incidence of traffic accidents, increasing efficiency and providing increased accessibility to those unable to drive [69], [35].

Several commercial companies currently have autonomous systems in operation:

- Waymo, a subsidiary of Alphabet Inc., operates at Level 4 and has deployed autonomous cabs in Phoenix, San Francisco and Los Angeles, operating without safety drivers onboard [50].
- Cruise, a subsidiary of General Motors, is currently operating Level 4 autonomous vehicles in San Francisco during daytime [23].
- Mercedes, in the US, has achieved the status of the first car company to reach certified Level 3 autonomous capabilities [16].
- BMW has confirmed that a Level 3 self-driving system will be launched in 2023 [47].

Autonomous vehicles have the potential to greatly reduce the incidence of traffic accidents. Human error is the cause of the vast majority of traffic accidents. Autonomous vehicles, being governed by algorithms and AI, do not get tired, distracted, or impaired and thus hold the promise of making our roads safer [41].

It is also anticipated that autonomous vehicles could improve traffic efficiency. With their advanced sensors and connectivity capabilities, autonomous vehicles can potentially communicate with each other and traffic management systems to optimize traffic flow, reducing congestion and improving overall travel efficiency [39], [24].

Moreover, the rise of autonomous vehicles could lead to new business models in the automotive industry, particularly in the form of Mobility-as-a-Service (MaaS). MaaS refers to a shift away from personally-owned vehicles towards a service where transportation is consumed as a service. This is already evident with the rise of ride-hailing and ride-sharing services and the development of autonomous vehicles could further accelerate this trend [26].

Despite the immense potential and progress made so far, significant challenges remain. These include technical challenges such as improving the robustness of perception systems, dealing with complex traffic situations and ensuring cybersecurity. There are also non-technical challenges, including legal and regulatory issues, societal acceptance and ethical considerations in the decision-making algorithms of autonomous vehicles [35].

The ongoing research and development efforts by various stakeholders, including automotive manufacturers, technology companies, research institutions and regulatory bodies, are crucial for overcoming these challenges and realizing the full potential of autonomous vehicles. As such, this is an exciting and dynamic field that is poised to reshape our transportation systems and societies in the years to come.

SLAM in Autonomous Cars

SLAM processes have to address several key challenges, one of which relates to the type of prior knowledge utilized for localization. Discussions suggest that High Accuracy Digital (HAD) maps might provide strong enough prior knowledge for these algorithms. However, the usefulness of these maps in localization processes remains to be fully verified. Furthermore, the placement of sensors on autonomous vehicles can significantly impact the performance of SLAM systems. For example, a laser sensor placed on the roof of the car is less likely to be affected by mobile obstacles compared to one placed in the undercarriage. The minimum necessary sensor set for localization remains undefined, yet it appears that laser scanners and cameras are favoured [8].

The trend of current SLAM systems is tending towards being lightweight and promoting multi-agent cooperation [13]. This allows application on low-powered hardware like embedded devices and considers multi-sensor fusion algorithms as the core of SLAM application in autonomous vehicles. The combination of SLAM with autonomous driving requires further exploration, particularly in real-world urban road environments, as current datasets often deviate from real-world performance [13].

Finally, safety in localization algorithms is paramount. Autonomous driving systems must design strategies to safely switch between multiple sources and account for potential failures and their impacts on the localization system. Despite these challenges, advancements in SLAM technologies and the increasing public recognition towards autonomous driving vehicles, coupled with high-performance mobile computing, point towards the potential for more practical applications of visual SLAM in the near future [13].

2.1.2 Autonomous Space Robots

Autonomous space robots are engineered to conduct scientific research, perform maintenance tasks and explore celestial bodies such as planets, moons and asteroids, particularly in environments where human intervention is challenging or infeasible [19], [40]. This category of robots includes orbiters, landers, rovers and satellites.

The development of autonomous space robots originated in the 1960s and 1970s when the first robotic probes were launched to explore the Moon, Mars and other celestial bodies. Notable early examples of robotic spacecraft include the Soviet Union's Luna and Mars missions and the United States Surveyor and Viking missions, all designed to gather information about extraterrestrial surfaces [53].

While early space robots primarily relied on remote control or preprogramming, the necessity for autonomy grew as researchers acknowledged the challenges and communication delays inherent in operating robots in deep space. The 1980s and 1990s marked a transition towards the development of increasingly autonomous space robots, capable of decision-making and navigating complex environments with minimal human intervention. Since then, autonomous space robots have become an integral component of planetary exploration, with progressively sophisticated systems deployed in missions to Mars, the Moon and beyond [66].

Modern autonomous space robots have significantly evolved over time. Advancements include Reinforcement Learning (RL) to improve task performance in satellite servicing or debris removal [67], enhancements in Guidance, Navigation, and Control (GNC) for in-orbit robotic missions [38] and the optimization of dual-arm space robot configurations to boost capture efficiency, stability and safety during on-orbit operations [68]. Collectively, these improvements demonstrate the ongoing progress in the development of modern autonomous space robots.

In July 2020, Perseverance was launched by NASA and successfully landed on Mars in February 2021 [42]. As part of the Mars 2020 mission, Perseverance's primary objective is to search for signs of ancient microbial life, study the planet's climate and geology and collect samples for a future Mars Sample Return mission [43]. Equipped with advanced instrumentation and technology, Perseverance showcases a high level of autonomy in terms of navigation, scientific sampling and decision-making. One of the rover's features is the Mars Helicopter, Ingenuity, which is a technology demonstration aimed at testing powered flight on another planet for the first time [44]. The Perseverance rover and Ingenuity helicopter exemplify the current state-of-the-art in autonomous space robots, paving the way for more advanced and capable missions in the years to come.

SLAM in Space Robots

One of the early implementations of a Rao-Blackwellized Particle Filter (RBPF)-based SLAM algorithm used a vision-based sensor to facilitate maps incorporating large numbers of visual landmarks, with the aim of extending this work to full 6-DOF representations better suited for space vehicles and exploration over rough terrains. Potential challenges were identified such as managing rarely observed Scale-Invariant Feature Transform (SIFT) features that clutter the data structure and necessitate costly rebalancing [54].

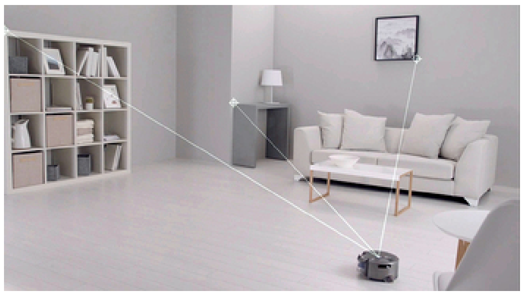
A more recent approach known as Globally Adapted (GA) SLAM, supports long-range low-supervision autonomous navigation by providing accurate relative localization, a local elevation map for navigation and a global pose correction method [20]. However, this system was designed with the constraints of limited onboard processing power, hence computational efficiency remained a significant challenge. Several improvements were proposed, including modelling the space of the problem in a discretized way and improving the quality of the local map by using a neighbourhood fusion method [20].

The development of datasets represents another significant challenge in this field [21]. A dataset, recorded on Mount Etna, Sicily, a lunar and Martian environment analogue, posed significant problems for SLAM due to visual aliasing and the absence of outstanding structural details. The dataset provides a tool for exposing limitations of traditional SLAM systems under these challenging conditions and motivates the development of novel localization and mapping approaches [21].

Despite the challenges, the continuous advancement and adaptation of SLAM algorithms for space robots demonstrate its potential for future applications in space exploration. The ongoing research in this area is focused on overcoming computational limitations, improving robustness and enhancing the adaptability of SLAM algorithms to extreme and complex environments.

2.1.3 Various Robots

Robotic platforms have also gained significant popularity among other fields in the private and also industrial sector. In 2022, the robotic market generated around 30 billion Swiss Francs in revenue, from which 25% comes from industrial robots, the other 75% from service robots, mostly used in private households [59]. Some well-known examples will be briefly described in this section.


(a) Dyson robot using vision system [2]

(b) Buddy Mobility with KYBURZ technology [29]

Figure 2.3: Autonomous Robots in action

Industrial robots

Robots in the industry can help humans and the economy in several aspects. Robots can achieve precise results with much fewer errors while running 24 hours a day. Some of the currently most used industrial robots are the following:

- Articulated Robots (Robotic arms): These robots have rotary joints that provide a high degree of freedom and flexibility. They resemble a human arm with multiple connected links. Articulated robots are widely used in assembly factories, material handling, welding and painting applications. Popular manufacturers of robotic arms are ABB, KUKA and Funac.
- SCARA Robots: SCARA stands for Selective Compliance Assembly Robot Arm. SCARA robots have a fixed vertical arm connected to a horizontal arm that can move in a circular motion. They are commonly used for tasks that require fast and precise horizontal movements, such as pick-and-place operations and assembly tasks.
- Cartesian Robots: Also known as gantry or linear robots, Cartesian robots have three linear axes (X, Y and Z) that move independently. They provide precise and synchronized movements in a rectangular coordinate system. Cartesian robots are often used in 3D printers or for tasks such as material handling, packaging and CNC machining.
- Delta Robots: Delta robots have a unique design with three or more arms connected to a common base. The arms are typically connected to parallelograms and driven by actuators. Delta robots are known for their high speed and precision, suitable for applications requiring fast and precise pick-and-place operations, such as packaging and sorting.
- Mobile Robots: Mobile robots are autonomous or semi-autonomous robots capable of moving and navigating on their own. They can be equipped with various features such as wheels, tracks, or even legs for mobility. Mobile robots are used for tasks such as material transportation and order picking in warehouses. Some commonly known manufacturers of mobile robots are Boston Dynamics,

ABB and Clearpath Robotics. There are also Swiss competitors in the field of mobile robotics like KYBURZ, manufacturing full-autonomous robots for postal delivery, as seen in Figure 2.3b.

Household helpers

Already since the mid-nineties, robot manufacturers are developing and selling household helper robots to make life more comfortable for end users. On one hand, companies like iRobot and Dyson manufacture vacuum cleaning robots, illustrated in Figure 2.3a, which are able to clean flat surfaces of different kinds autonomously. For around ten years, these robots are also running SLAM algorithms to create a digital map of their environment, sensed through different sensors like laser scanners, cameras or gyroscopes. Commonly, the robots hold an initially created map of the house while always improving it in order to react to changes in the map caused by moved objects.

On the other hand, companies like Husqvarna and Bosch produce robotic lawnmowers, very similar to vacuum cleaners. The first versions of these robots used physical wires limiting the space the robot should operate in. Further development led to improved robots being able to perform SLAM algorithms supported by Global Navigation Satellite System (GNSS) data to spatially localise itself. In order to further improve the localisation, they use the RTK (Real-Time Kinematics) technology which uses a fixed reference station close to the robot. The exact position of the station is known so the station is able to calculate the error of the estimated position of the GNSS system. This error will then be forwarded to the robot improving his own GNSS estimation significantly.

Both mentioned systems are able to automatically return to their charging stations and therefore require only minimal input from the owner.

2.2 Probabilistic Robotics

This section provides the main theoretical background for this thesis and is heavily inspired by the book *Probabilistic Robotics* [63]. Probabilistic techniques have emerged as a cornerstone for addressing the inherent uncertainty in robotic perception and decision-making.

2.2.1 Uncertainty in Robotics

Collected data are often imperfect, leading to discrepancies between the robot's perception of its environment and the true state of the environment. Understanding and addressing these uncertainties is crucial for developing robust systems that can effectively navigate and interact with their environments. The primary sources of uncertainty include:

- Sensor Noise: Sensors are subject to noise and measurement errors, which may result from manufacturing imperfections, environmental conditions, or other external factors. This noise can cause discrepancies between the robot's perception of its environment and the true state of the environment.
- Actuator Noise: Similar to sensors, robot actuators can also introduce noise and inaccuracies due to manufacturing imperfections or wear and tear. These inaccuracies can lead to discrepancies between the robot's intended and actual actions, further contributing to the overall uncertainty in the robot's state.

- Environment Complexity: Real-world environments are often complex and dynamic, presenting challenges for robots to perceive and model their surroundings accurately. Additionally, some aspects of the environment may be partially observable or hidden, further increasing the level of uncertainty.
- Modeling Errors: In robotics, mathematical models are often used to describe motion and measurement processes. These models are simplifications of real-world processes and can introduce errors and uncertainties due to the assumptions made during their development.

In light of these uncertainties, deterministic approaches often prove inadequate for state estimation and decision-making in robotics. Probabilistic methods provide a framework to cope with these uncertainties and allow robots to estimate their state and make informed decisions in complex and uncertain environments.

2.2.2 Robot Environment Interaction

In robotics, the interaction between a robot and its environment is essential for its ability to perceive and act effectively. An actor, which can be a human, thermostat, software agent, or robot, perceives its environment and acts upon it. Robots acquire information about their surroundings using sensors. However, a robot can only perceive aspects of the environment for which it has equipped sensors. In cases where a specific sensor is absent, the robot is unable to collect information regarding that particular parameter.

The collection of data that can potentially impact future decisions can be categorized into two types: environment measurement data, such as camera images or laser sensor data and control data, which refers to information about changes in the environment's state, like the velocity or frequency of the robot's wheel revolutions.

The state often includes variables such as the robot's pose, which encompasses its Cartesian coordinates and angular orientation. A robot cannot directly measure its pose so it derives the pose from the collected data instead. This results in the formation of a belief, which is a conditional probability distribution representing the robot's understanding of its current state. Control data, on the other hand, refers to the commands sent to the robot's actuators to execute specific actions, such as desired velocities, steering or joint angles. Probabilistic techniques are employed to deal with uncertainties in sensor data and actuation, allowing the robot to make informed decisions and navigate complex environments.

2.2.3 Robot Motion

As described above, a robot aims to derive its pose from the data about its environment and state. The state at a specific time t of a robot operating in a planar environment is described by Equation 2.1 where (x, y) represents the location of the robot (translational component) and θ the orientation of the robot (rotational component).

$$x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.1}$$

A motion model describes the transition a robot faces when changing its own state by actuators like the motor. Since actuators in practical applications are noisy, a direct kinematic approach to solve this task may not be suitable. In probabilistic robotics, the motion model is a distribution defined by Equation 2.2.

$$p(x_t|u_t, x_{t-1}) \tag{2.2}$$

The distribution represents the likelihood of the new robot state x_t, given a set of actions u_t by the controls at time t and x_{t-1}, representing the previous state before the controls influenced the robot.

Odometry Motion Model

In practice, robots commonly employ odometry data as the controls input for their motion models. This preference stems from the availability of odometry data in many commercial products and the inherent limitations in the accuracy of the robot's actuator actions, which cannot match the precision offered by odometry data.

The challenge with odometry data lies in the fact that the measured sensor data is always relative to the previous state of the sensors, such as gyroscopes. In an ideal scenario without any noise, the measured changes in sensor data would accurately reflect the corresponding changes in the robot's state in the physical world. However, due to the presence of sensor noise, this correspondence is not practically achieved. Nevertheless, by utilizing the difference in odometry data between time steps $t-1$ and t to estimate $\overline{x}_t$, as illustrated in Equation 2.2, a reasonably accurate approximation of the true state x_t can be obtained.

A further challenge in the use of sensor data in odometry-based motion models is the issue of sensor drift. Sensor drift, which may be caused by temperature changes, ageing, or other factors, can introduce a slow, accumulative error in the sensor readings over time. This drift can lead to significant discrepancies between the robot's estimated state $\bar{x}_t$ and its true state x_t, especially over extended periods of operation. Thus, strategies for mitigating the impact of sensor drift are an important aspect of effective robot motion modelling.

2.2.4 Bayes Filters

Bayes filters form the foundation for recursively estimating the state of dynamic systems subject to uncertainty. Employing the principles of Bayesian inference, these filters update the probability distribution over the state space based on incoming sensor measurements. They fuse prior knowledge with current observations to derive a posterior belief about the system's state. Central to the Bayes filter is the prediction-update cycle, which encompasses two steps:

- Prediction: Predicts the system's state based on its motion model, accounting for the uncertainty in the control input. The motion model, denoted by $p(x_t|u_t, x_{t-1})$, captures the relationship between consecutive states, given the control data and generates a predicted belief about the next state, $\overline{bel}(x_t)$, before incorporating sensor measurements.

- Update: Updates the belief by incorporating new sensor measurements through the measurement model, denoted by $p(z_t|x_t)$, which describes the likelihood of obtaining specific measurements given the current state. The update fuses the predicted belief with the latest measurements, producing an updated posterior belief about the system's state, $bel(x_t)$.

By iteratively performing this cycle, Bayes filters enable the robot to maintain an up-to-date belief about its state and the environment, facilitating informed decision-making and navigation despite inherent uncertainties. The versatility of Bayes filters allows for various implementations, such as Gaussian filters, particle filters and grid-based filters, to accommodate different assumptions about the state and measurement models. The prediction and update steps can be represented with Equation 2.3 where η is a normalizing constant.

$$\begin{aligned} \overline{bel}(x_t) &= \int p(x_t|u_t, x_{t-1}) \cdot bel(x_{t-1}) dx_{t-1} \quad &\text{(Prediction)} \\ bel(x_t) &= \eta \cdot p(z_t|x_t) \cdot \overline{bel}(x_t) \quad &\text{(Update)} \end{aligned} \tag{2.3}$$

A state can be predicted with the information of the state before and the current controls. The state of the robot defines what a robot measures. Figure 2.4 illustrates the dependencies between the state x, control u and measurement z. The state x_t is stochastically dependent on the state x_{t-1} and the control u_t, while the measurement z_t depends stochastically on the state x_t. These dependencies capture the evolution of states and measurements, reflecting the underlying dynamics of the robot and its environment.

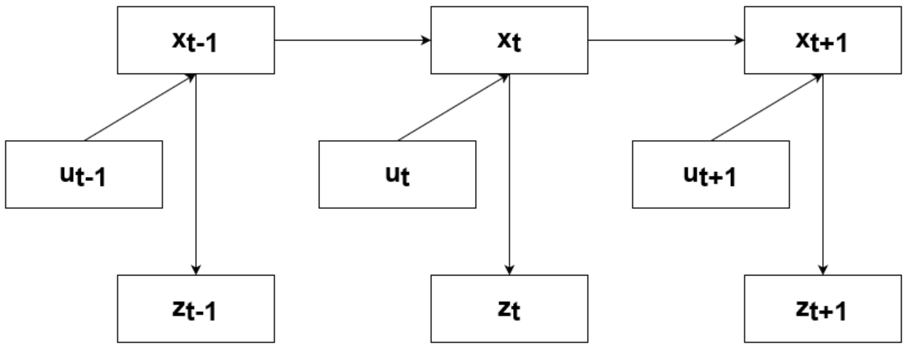


Figure 2.4: The state is dependent on the state before and the control. The measurement depends on the state. [63]

2.2.5 Gaussian Filters

Gaussian filters represent a specific class of Bayes filters that assume the underlying state and measurement models follow Gaussian distributions. By exploiting the properties of Gaussian distributions, such as closure under linear transformations and the convolution operation, these filters simplify the filtering process, resulting in computationally efficient algorithms. Closure under linear transformations means that the result of a linear transformation applied to a Gaussian-distributed variable remains Gaussian-distributed. The convolution operation refers to the process of combining two functions to produce a third function that represents the overlap between the original functions. In Gaussian filters, this property guarantees that the product of two Gaussian-distributed variables remains Gaussian-distributed. Gaussian filters are particularly well-suited for systems with linear dynamics and Gaussian noise, as they provide a compact and analytically tractable representation of the state's probability distribution.

Compared to generic Bayes filters, Gaussian filters make specific assumptions about the state and measurement models to derive closed-form solutions for state estimation. While Bayes filters can accommodate various representations of uncertainty, Gaussian filters rely on the Gaussian distribution to model the uncertainties in both the robot's state and sensor measurements. They can be described by the mean μ and the covariance Σ as illustrated in Equation 2.4.

$$\begin{aligned} \mu &= \int x \cdot bel(x)dx && \text{(Mean)} \\ \Sigma &= \int (x-\mu)(x-\mu)^T bel(x)dx && \text{(Covariance)} \end{aligned} \tag{2.4}$$

The two primary types of Gaussian filters employed in robotics are the Kalman filter and the Extended Kalman Filter (EKF). The Kalman filter is designed for systems with linear state transition and measurement models, while the EKF approximates nonlinear models by linearization using the first-order Taylor series expansion. Although both filters have limitations when dealing with highly nonlinear systems, Gaussian filters remain prevalent in robotics due to their computational efficiency, ease of implementation and ability to provide a closed-form solution for the state estimation problem.

Kalman Filter

The Kalman filter is a widely-used Gaussian filter employed in robotics and other applications involving state estimation of linear systems with Gaussian noise. It is a recursive data processing algorithm that combines a system's dynamic model with incoming sensor measurements to estimate the state of a dynamic system in real-time, mathematically represented in Equation 2.5. The Kalman filter operates by predicting the system's state based on the motion model and updating this prediction with new measurements, leveraging the Gaussian distribution's properties to maintain a compact and tractable representation of the state's probability distribution.

For the Kalman filter, the prediction and update steps are as follows:

- Prediction: Predicts the next state $\overline{\mu_t}$ and its covariance $\overline{\Sigma_t}$, using the linear motion model, considering the control input u_t and associated noise. It propagates the current state's mean and covariance forward in time, accounting for the uncertainty introduced by the system dynamics and control noise, represented by R_t.
- Update: The filter computes the Kalman gain K_t, which weighs the importance of the new measurement z_t relative to the predicted state μ_t. The gain is used to update the predicted state's mean and covariance, incorporating the latest sensor measurement z_t and its associated noise Q_t to produce a refined state estimate μ_t and Σ_t.

The equations involved in these steps are as follows:

$$\begin{aligned} \overline{\mu_t} &= A_t\mu_{t-1} + B_t u_t && \text{(Prediction of the mean)} \\ \overline{\Sigma_t} &= A_t\Sigma_{t-1}A_t^T + R_t && \text{(Prediction of the covariance)} \\ K_t &= \overline{\Sigma_t}C_t^T(C_t\overline{\Sigma_t}C_t^T + Q_t)^{-1} && \text{(Kalman gain computation)} \\ \mu_t &= \overline{\mu_t} + K_t(z_t - C_t\overline{\mu_t}) && \text{(Update of the mean)} \\ \Sigma_t &= (I - K_tC_t)\overline{\Sigma_t} && \text{(Update of the covariance)} \end{aligned} \tag{2.5}$$

In the equations, A_t and B_t represent matrices of the size $n \times n$ for A and $n \times m$ for B, where n is the dimension of the state vector x_t and m is the dimension of the control vector u_t. Using these matrices, the equation becomes linear in its arguments. C_t is a matrix of the size $k \times n$, where k is the dimension of the measurement z_t representing the Jacobian of the measurement [63].

Due to its computational efficiency and optimal performance under linear Gaussian assumptions, the Kalman filter is widely employed in various robotic applications, including sensor fusion, navigation and control.

Extended Kalman Filter

The EKF is an extension of the Kalman filter designed to handle mildly nonlinear systems. It addresses the limitations of the Kalman filter in dealing with nonlinear motion and measurement models by linearizing these models using first-order Taylor expansions, which approximate the nonlinear functions with locally-linearized versions. The mathematical representation can be seen in Equation 2.6.

The EKF algorithm follows the same prediction-update cycle as the Kalman filter, but with modifications to accommodate the nonlinearities:

- Prediction: Predicts the next state $\overline{\mu_t}$ and its covariance $\overline{\Sigma_t}$, using the nonlinear motion model $g(u_t, \mu_{t-1})$ and its linearized version, known as the Jacobian matrix G_t, considering the control input u_t and associated noise R_t. It propagates the current state's mean and covariance forward in time, accounting for the uncertainty introduced by the system dynamics and control noise.
- Update: Computes the Kalman gain K_t, similar to the Kalman filter, but with the measurement model's Jacobian matrix H_t to account for the nonlinearity. The gain is used to update the predicted state's mean and covariance, incorporating the latest sensor measurement z_t and its associated noise Q_t to produce a refined state estimate μ_t and Σ_t.

The equations involved in these steps are as follows:

$$\begin{aligned}
\overline{\mu_t} &= g(u_t, \mu_{t-1}) && \text{(Prediction of the mean)} \\
\overline{\Sigma_t} &= G_t \Sigma_{t-1} G_t^T + R_t && \text{(Prediction of the covariance)} \\
K_t &= \overline{\Sigma_t} H_t^T (H_t \overline{\Sigma_t} H_t^T + Q_t)^{-1} && \text{(Kalman gain computation)} \\
\mu_t &= \overline{\mu_t} + K_t(z_t - h(\overline{\mu_t})) && \text{(Update of the mean)} \\
\Sigma_t &= (I - K_t H_t)\overline{\Sigma_t} && \text{(Update of the covariance)}
\end{aligned} \tag{2.6}$$

2.2.6 Particle Filter

Besides the Gaussian filters there are particle filters, also known as Sequential Monte Carlo methods, used in probabilistic robotics for tracking and prediction, in the presence of noise and uncertainty. The primary advantage of particle filters over other types of filtering methods, such as Kalman filters, is their ability to handle non-linear and non-Gaussian systems. Given a system's model and some initial conditions, particle filters estimate the state of a system at a given point in time by using a set of particles to represent the Probability Density Function (PDF) of the system's state.

A particle filter uses a set of particles, where each particle is represented as x_i, w_i. Each particle is a hypothesis about the state and its weight represents the likelihood of that hypothesis. x_i represents the state of the system and w_i denotes the weight of the particle, which signifies the particle's importance.

The fundamental steps in the particle filter algorithm can be summarized as follows:

- Initialization: Initially, a set of particles are generated and these are usually drawn from the prior distribution of the state.
- Importance Sampling: For every time step, each particle's state is predicted using the system model and the predicted state is then compared with the actual observed data. The difference between the predicted state and the observed data is used to compute the weight of each particle, which essentially denotes the likelihood of the state.
- Resampling: Once the weights are calculated for all particles, a resampling step is performed. In this step, particles with higher weights are more likely to be selected, while those with lower weights may disappear. This forms the new particle set for the next time step.

This process is expressed using Bayes' theorem and concepts of probability distributions. For instance, the importance sampling step can be represented with Equation 2.7.

$$w_i = p(z_t | x_i^t) \tag{2.7}$$

Here, z_t is the observed data at time t and x_i^t is the state of particle i at time t. Equation 2.7 states that the weight of a particle is proportional to the likelihood of the observed data given the particle's state.

2.3 Simultaneous Localization and Mapping

Simultaneously localizing a robot within a known environment and creating a map of its surroundings are manageable tasks when the exact spatial position of the robot is consistently known. However, placing a robot in an unknown environment and requiring it to both generate a map and localize itself within this map solely based on control inputs u and observations of landmarks z significantly increases the complexity. This scenario introduces the SLAM problem.

There are two ways to approach the SLAM problem. Using an online SLAM approach, the robot's new state x_t, as well as the updated map m, is described by the distribution $p(x_t, m|z_{1:t}, u_{1:t})$ which only uses all measured observations of landmarks $z_{1:t}$ as well as all control data $u_{1:t}$ to estimate the latest state x_t of the robot. Online SLAM does not care about past state estimations.

The second way of solving the SLAM problem is full SLAM, which involves processing the measured observations and stored controls in a batch after they have been recorded. This leads to the only slightly different looking distribution $p(x_{1:t}, m|z_{1:t}, u_{1:t}$. Due to this fact, full SLAM approaches provide more accuracy, whereas online SLAM approaches can perform better in real-time applications since they directly process incoming sensors and control data.

2.3.1 EKF SLAM

The EKF SLAM emerged as one of the pioneering solutions to the SLAM problem. Utilizing the EKF, this approach effectively linearizes the inherently nonlinear motion and measurement models prevalent in robotics, thereby enabling more efficient map building and localization. EKF SLAM functions based on a prediction-update cycle as explained in Section 2.2.5, aiming to resolve the online SLAM problem in real-time scenarios.

The EKF SLAM algorithm unfolds in two sequential phases: the prediction phase and the update phase.

- **Prediction Phase:** The algorithm starts by estimating the robot's current position and orientation based on its motion model. Using the most recent control inputs, it projects the robot's state into the future, simultaneously updating the covariance matrix E, a $n \times n$ representation of mutual uncertainties between all n landmarks.
- **Update Phase:** Following prediction, the algorithm integrates new sensor data into the model. It first formulates an expected measurement based on the current state estimate. After computing the measurement residual, the difference between the actual and expected measurements, it updates the state estimate and the covariance matrix in accordance with the Kalman gain. The updated state estimate and covariance matrix encapsulate the revised belief about the robot's state and associated uncertainty.

In environments with many landmarks, the iterative update of the covariance matrix E at each step becomes a computationally challenging task. This computational burden can exceed the often limited processing capabilities of a robot, particularly during extended operations.

Given these constraints, EKF SLAM is generally favoured in environments with relatively few landmarks. It is typically utilized in scenarios with fewer than 1'000 landmarks due to its scalability limitations, striking a balance between computational efficiency and the fidelity of landmark tracking and estimation.

2.3.2 FastSLAM

FastSLAM is an efficient solution to the SLAM problem that utilizes a particle filter approach. This allows it to manage a large number of landmarks effectively. Each particle within the filter represents a hypothesis concerning the robot's trajectory and concurrently, each maintains an individual map of the environment. The individual maps typically consist of various features or landmarks, with each having its associated position and uncertainty.

The FastSLAM algorithm works as follows:

- **Prediction:** Similar to the particle filter, the first step is to predict the state of each particle based on the motion model of the robot and the most recent control inputs.
- **Update:** When a new observation arrives, the weights of the particles are updated based on the likelihood of this observation given the predicted state and map of each particle. Each particle also updates its map based on the new observation.
- **Resampling:** Finally, the particles are resampled based on their weights.

In FastSLAM, each particle maintains an individual map. These maps are typically represented as a set of features or landmarks, where each landmark j is modeled by a Gaussian distribution with a mean μ_j and a covariance Σ_j.

When a new observation arrives, the map within each particle is updated based on the observation. Specifically, if the observation corresponds to a known landmark, the position and covariance of the landmark are updated. If the observation corresponds to a new landmark, a new landmark with its position and covariance is added to the map.

The update of a landmark's position μ_j and covariance Σ_j based on a new observation z can be expressed with Equation 2.8 where K is the Kalman gain, $h(\mu_j)$ is the predicted observation based on the landmark's position, H is the Jacobian of h with respect to μ_j and I is the identity matrix.

$$\begin{aligned}\mu_j &= \mu_j + K \cdot (z - h(\mu_j)) \\ \Sigma_j &= (I - K \cdot H) \cdot \Sigma_j\end{aligned} \tag{2.8}$$

This performs a Kalman update on the landmark's position and covariance.

2.3.3 Advantages and Limitations

EKF SLAM's strength lies in its ability to effectively linearize typically nonlinear motion and measurement models, allowing efficient map-building and localization. It is highly suitable for environments with relatively few landmarks due to its computational demands. However, its computational intensity increases exponentially with the number of landmarks, limiting its scalability. It also assumes a Gaussian noise model and linear system dynamics, which might not be accurate for all environments and robots.

FastSLAM excels in computational efficiency and scalability, managing a large number of landmarks effectively. Its particle filter approach allows for handling nonlinear system dynamics and non-Gaussian noise. Despite its benefits, FastSLAM relies heavily on known data associations, requiring accurate identification of corresponding observations and landmarks. It also assumes independence between robot motion and landmark observations, which might not always hold in real-world scenarios.

In conclusion, EKF SLAM and FastSLAM are valuable tools in probabilistic robotics, offering solutions to complex problems of tracking, prediction and mapping in uncertain and dynamic environments. They find applications across various fields, including autonomous vehicles, mobile robotics and navigation systems.

2.3.4 Landmark correspondence detection

All previously discussed SLAM algorithms predict where a detected cone is located spatially. However, all algorithms need functionality to determine if a detected cone is one which was already detected and therefore is already available in the stored map, or alternatively is a new cone seen for the first time.

A possible approach to solve this problem uses the Mahalanobis distance, with which it gets calculated, how many standard deviations away a point $\vec{x}$ is from the center $\vec{\mu}$ of a distribution Q as described with Equation 2.9 where S is the covariance matrix of the known points.

$$d_M(\vec{x}, Q) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})} \tag{2.9}$$

As can be seen, the Mahalanobis distance takes the covariance of the distribution into account. This improves the decision of correspondence significantly compared to simple distance thresholds. Put in other words, the Mahalanobis distance can be understood as the distance of the point $\vec{x}$ to $\vec{\mu}$, divided by the width of the ellipsoid representing the covariance of the distribution. A visual representation of the Mahalanobis distance can be seen in Figure 2.5.

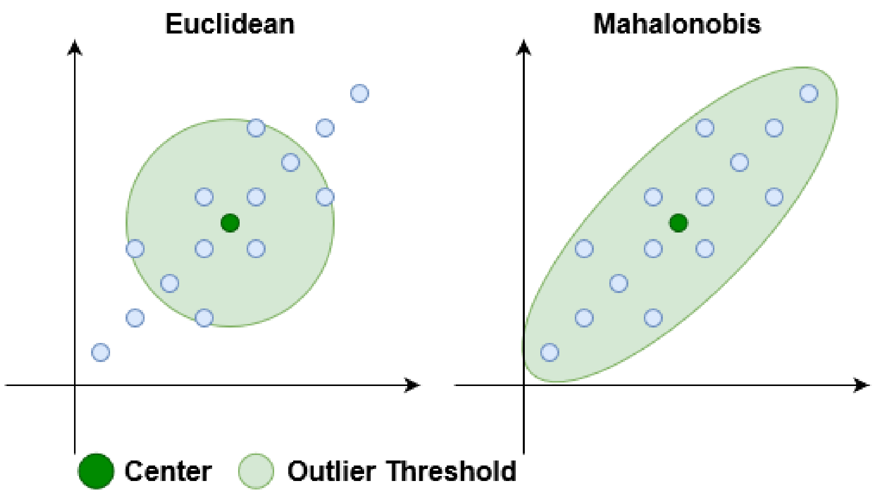


Figure 2.5: Difference between Euclidean distance and Mahalanobis distance [10]

2.4 End-to-End Deep Learning Approaches

The field of AI is transitioning from addressing specialized, deterministic problems to contending with broader, more complex problems. This shift from narrow AI to artificial general intelligence has been largely enabled by the advent of deep learning techniques [32]. End-to-end deep learning represents a paradigm where a complex system is learned as a single unified function, mapping raw inputs to desired outputs without explicitly designed intermediate steps or representations. This approach, powered by deep neural networks capacity to encapsulate relationships and dependencies within their hidden layers, has found widespread adoption across various domains such as natural language processing [1], image recognition [34] and autonomous driving [62].

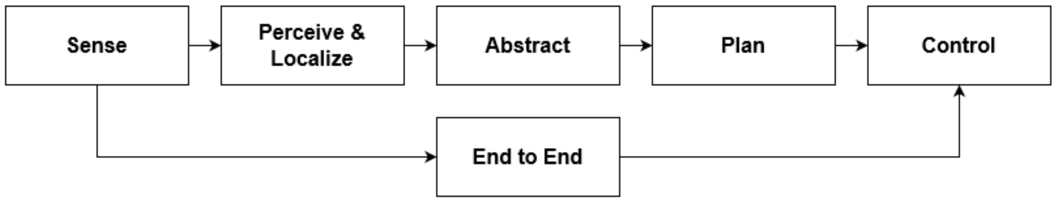


Figure 2.6: In end-to-end systems a single unified function is used instead of splitting the problem up into sub-tasks [62]

In autonomous driving, end-to-end learning is capable of directly comprehending the mapping from raw sensor data to vehicle control commands, circumventing the need for explicitly programmed intermediary steps, like object detection or path planning. This concept is illustrated in Figure 2.6. It is possible with a convolutional neural network to map raw pixel inputs from car sensors to steering commands. However, to apply the end-to-end approach with sufficient robustness to consumer products further improvements have to be made [7].

2.4.1 Reinforcement Learning from Human Feedback

Supervised learning demands plenty of training data. While humans can be observed to generate training data for autonomous systems there are shortcomings. Humans do make minor and severe errors while driving. Besides supervised learning, it is possible to use reinforced learning where the desired behaviour is rewarded and the agent can learn by improving itself. Nevertheless, it is hard for complex environments to define an effectively performing reward function. A system should therefore not learn to imitate a human driver but rather learn the human values in driving behaviour and build a reward function according to these values.

Reinforcement Learning from Human Feedback (RLHF) is a concept that solves this problem by learning a reward model for a specific task based on human feedback. It then trains a policy to optimize the reward received from this reward model. A significant benefit of RLHF is the sample efficiency required to train the reward model. This allows the system to strike a balance between learning from human feedback and generalizing from the task data. The ultimate goal of RLHF is to create a reward model that represents human preferences for how a task should be done, which is also known as Inverse Reinforcement Learning. While these principles are currently applied predominantly to Large Language Modelss (LLMs), their application could also be promising in the field of autonomous driving [30], [14], [60].

2.4.2 Parsimony and Self Consistency

Autonomous intelligent agents ideally have capabilities to reflect their past experiences and the current environment leading to an internal world model. Neuroscience suggests that the world model of the human brain is highly structured anatomically and functionally [46], [11], [5]. This structure is believed to be the key to efficient decision-making [27]. On the other hand, the current brute-force end-to-end training of black-box models contradicts this structural concept. Resulting in enormous model sizes and varied problems such as the lack of richness in final learned representations due to neural collapse [48], lack of stability in training due to mode collapse [58], lack of adaptiveness and susceptibility to catastrophic forgetting [36] and lack of robustness to deformations [4] or adversarial attacks [61]. To overcome these limitations intelligence has to be understood in a more principled and unifying perspective incorporating the functional and organizational structure of natural intelligent systems. This can be achieved with the two fundamental principles of parsimony and self-consistency [34].

Parsimony answers the question of what to learn. The principle stresses the importance of learning the most critical and representative features from the data, promoting the principle of Occam's razor within the learning process. This results in low-dimensional structures in a compressed, linear and independent form. But Parsimony alone does not ensure a learned model with the mentioned characteristics will capture all important information in the data sensed about the external world. To ensure that the internal representation of the real-world data matches the actual data the principle of self-consistency is used [34].

Self-consistency aims to create a model that best mirrors the external world by minimizing the discrepancy between observed and regenerated input. This is achieved by generating input data for the system with the compressed representation. The generated data is then encoded again to a compressed representation. The original representation and the new representation can be compared. These steps can be repeated while minimizing the rate reduction for the decoder and maximizing the rate reduction for the encoder leading to a self-correcting closed-loop system with an increasingly better-fitting representation of the input data. The system is illustrated with the example of handwritten digits in Figure 2.7 [34].

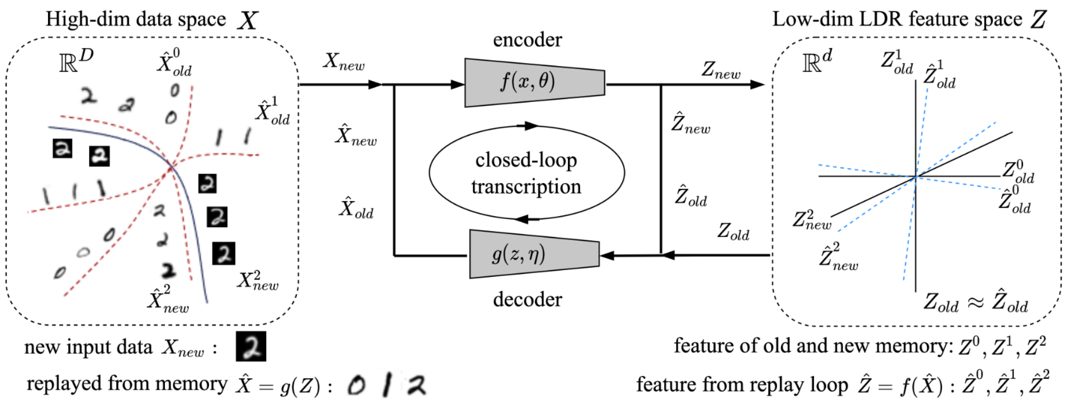


Figure 2.7: Incremental learning via a compressive closed-loop transcription [34]

Although divide and conquer has long been a cherished tenet in scientific research, when it comes to understanding a complex system such as Intelligence, this unite and build approach could lead to new possibilities [34].

2.4.3 Specialized to General

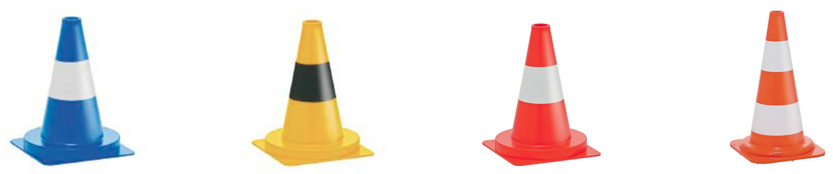
In conclusion, the shift towards end-to-end deep learning still has to overcome problems. There are promising principles like RLHF, parsimony and self-consistency, offering approaches towards more general problem-solving capabilities. These new paradigms could help in developing more autonomous, reliable and effective AI systems.

2.5 Formula Student

Formula Student teams around the globe have been developing and implementing various SLAM algorithms for their driverless vehicles, with the ultimate aim to improve performance in the competition's autonomous racing tasks. In this section, we will take a look at the race's procedure and rules, the work done by selected teams, including KA-RaceIng, Academic Motorsports Club Zurich (AMZ) and the work done by our predecessor in the context of SLAM.

2.5.1 Races

Each country participating in FS organizes one race in the summer after the spring semester. The races usually last around one week and consist of static and dynamic tests. During the static tests, the business plan, manufacturing and costs as well as the engineering design itself will be assessed by external experts from the host country. For each of the mentioned disciplines, points will be awarded and added to the overall score of the team. Dynamic tests include different kinds of race courses designed to test the physical and electrical capabilities of the car. The courses are limited by cones shown in Figure 2.8. The autonomous software of the race cars is challenged to navigate the course using camera vision detecting these cones with the help of various sensors.



(a) Blue cone used to limit courses on the left

(b) Yellow cone used to limit courses on the right

(c) Small red cone used to mark stop area

(d) Large red cone used to mark start and finish line

Figure 2.8: Official cones used in FS races [57]

2.5.2 KA-RaceIng

The Karlsruhe Institute of Technology (KIT) Formula Student team has explored the use of the two SLAM algorithms EKF SLAM and GraphSLAM in their autonomous race car. Their research focused on comparing the performance of these two algorithms, examining factors like accuracy, efficiency and computational load [31].

KIT's performance analysis concluded that GraphSLAM outperformed EKF SLAM in terms of accuracy, with their 2020 EKF SLAM version showing a considerable improvement over its 2019 version. However, it was noted that the accuracy of EKF SLAM 2020 dropped significantly over the course of a lap and was only corrected later. This suggests that while it could provide reasonably accurate maps, it may struggle with localization during the run. On the other hand, GraphSLAM continuously delivered highly accurate results [31].

The improved accuracy of GraphSLAM, however, came at a cost. It required more computational resources, with increased CPU load noticeable across different racing disciplines. To address this, the team suggested the use of a sliding window in the Trackdrive discipline, which can reduce the overall CPU load. Despite the increased load, initial tests on the vehicle showed sufficiently available processing power, pointing to the feasibility of using GraphSLAM in their autonomous car [31].

2.5.3 Academic Motorsports Club Zurich

The AMZ Driverless provided a very valuable paper describing their autonomous system in 2019. The lack of distinguishable landmarks, apart from the cones described by their position and colour, calls for an algorithm capable of handling uncertain data association. The SLAM algorithm used by AMZ Driverless is FastSLAM 2.0. The nature of this particle filter algorithm inherently provides multi-hypothesis data associations and its performance can be conveniently traded-off against runtime by adjusting the number of particles. Moreover, FastSLAM exhibits linear scalability with the number of landmarks, outperforming the quadratic complexity of an EKF SLAM algorithm in computational performance [28].

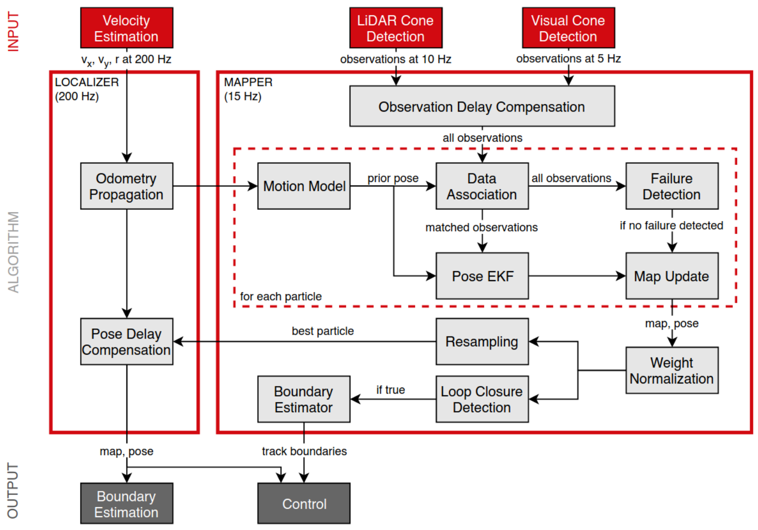


Figure 2.9: Detailed SLAM architecture used to fuse landmark observations from camera and LiDAR with velocity estimates into a coherent map and pose within the map. The dashed line visually demonstrates which parts of the algorithm are computed on a particle basis. [28]

The architecture of their SLAM system, illustrated in Figure 2.9, is divided into two parts: a localizer, which processes velocities at 200 Hz and a mapping algorithm that combines landmark observations from both perception pipelines and the car's velocity estimate for a high-frequency map update. Additionally, the system implements a two-stage sensor failure detection to utilize the redundant sensor setup. This process aims to avoid irreversible map updates caused by malfunctioning sensors by penalizing measurements that contradict previous observations [28].

When it comes to lap closure detection and post-processing, AMZ's approach assumes a static map after the completion of one lap, at which point the quality of the map will not significantly improve with additional observations. The car actively detects the completion of the first lap, based on a set of rules including the standard deviation of the position, the car's heading and proximity to the starting point. This detection does not globally correct the map, as global consistency is already achieved through FastSLAM's resampling [28].

Upon the completion of the first lap, AMZ's approach switches from SLAM to Localization mode. The track boundaries are identified and the map ceases to update. The map corresponding to the most likely particle (highest weight) is used for subsequent laps. Localization in the created map is achieved using Monte Carlo localization, computing the mean over all particles to ensure a smooth pose update [28].

2.5.4 Zurich UAS Racing

The ZUR team is participating in a FS DC for the first time this year. Despite this, the driverless automation team was established early on and has already carried out substantial work that the team can leverage. For the localization and mapping problem, two projects have been conducted: initially testing multiple methods for localization and subsequently improving the localization system and investigating the accuracy of pose tracking and the GPS sensor.

Several methods for localization were tested and implemented within the ROS2 framework for the overall system. Pose tracking was examined with multiple setups and tailored for this specific application. However, the GPS for the Kalman Filter and EKF could only be tested with simulated data. The team recommended further testing, involving the simultaneous recording of ZED2i data with GPS data in Rosbags. This would help verify and optimize the functionality of the Kalman Filter algorithm and GPS coordinate conversion using actual data. Ideally, a testing environment that closely mimics the race conditions and whose ground truth is known, should be set up for result analysis. The team also suggested incorporating additional sensors like an external IMU, LiDAR and wheel encoders and potentially augmenting the localization module with a mapping component, transforming it into a SLAM module.

Building on this information, a subsequent project was launched. The team focused on identifying and rectifying weaknesses in the localization system and developed a concept for implementing SLAM. The software architecture of the localization system was optimized. Additionally, the team investigated the accuracy of pose tracking and GPS. They discovered that the measurement error of the GPS can significantly impact the result when fused with other sensors. Therefore, it would be ideal in the future to correct GPS with other sensors and avoid heavily weighting the GPS sensor.

Chapter 3

Methodology

3.1 Overview

The project methodology, as outlined in Figure 3.1, is designed with a network of inter-dependencies and bidirectional relationships connecting the various stages of the project. This systematic approach ensures that each component and decision is firmly rooted in the essential research, planning and requirements.

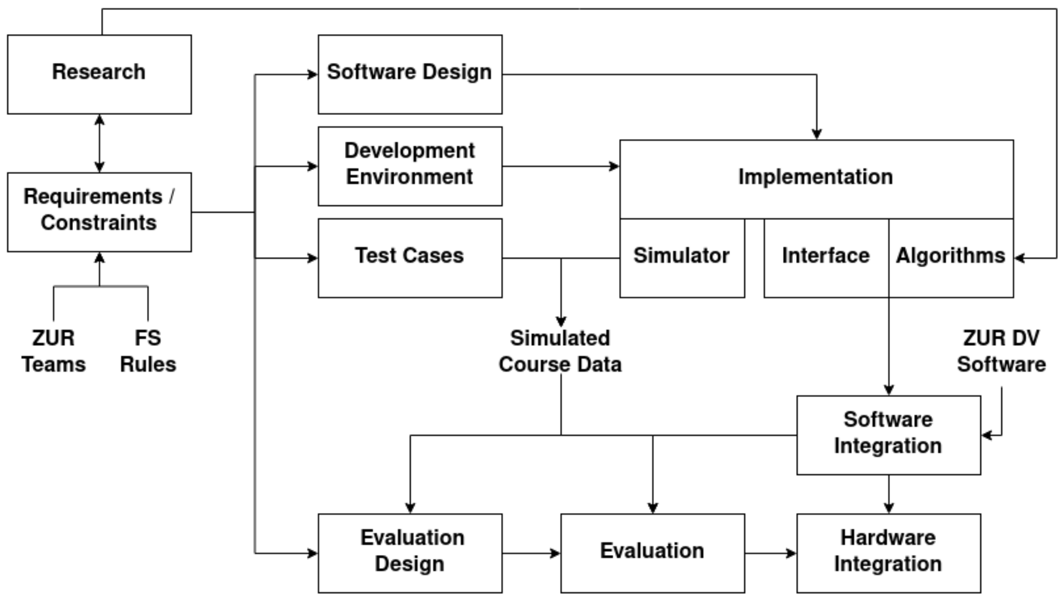


Figure 3.1: Division of tasks for the successful implementation of a SLAM algorithm

3.1.1 Requirements and Constraints

The inception of this thesis involves defining the project's Requirements and Constraints, which are influenced by the rules of the FS DV competition and the requirements of the autonomous system groups. These guidelines not only steer the direction of the research but also lay the groundwork for the software design, development environment, test cases and evaluation design.

3.1.2 Research

With the framework established, the next phase is Research. This exploration both informs and is informed by the project's Requirements and Constraints. It also directly contributes to the development of the SLAM algorithms utilized in the project, ensuring their alignment with best practices.

3.1.3 Software

The Software Design serves as a blueprint for the Implementation stage, which is further facilitated by the previously determined Development Environment. During the implementation, the simulator, the interfaces for other teams and the SLAM algorithms derived from the Research are realized.

3.1.4 Test Cases

The test cases involve assessments with simulated course data. The simulator is employed to generate the simulated course data, including simulated sensor data.

3.1.5 Evaluation

In the evaluation phase, the software system's performance is assessed using both course data and simulated course data. The evaluation metrics and criteria are defined within the evaluation design. This phase offers insights into the system's preparedness for hardware integration.

3.1.6 Integration

This process encompasses integrating the SLAM algorithm into the main repository of the autonomous system team. Successful Software Integration paves the way for the Hardware Integration phase, wherein the software is deployed onto the autonomous vehicle.

3.2 Requirements and Constraints

The successful execution and implementation of a project largely depend on its requirements and constraints, as they provide the foundation on which the project activities and resources are planned and executed. In this section, the critical requirements and constraints for the development and implementation of multiple SLAM algorithms in the autonomous driving system are discussed.

3.2.1 Requirements

The project aims to develop a robust and effective solution for the SLAM problem in autonomous driving. The requirements that guide the development and assessment of the SLAM algorithms are:

- The development and implementation of multiple working SLAM algorithms that suit the FS DC context.
- The definition of a suitable development environment to facilitate the implementation.
- The development of metrics to evaluate the performance and effectiveness of the implemented SLAM algorithms.
- The design and definition of test cases to validate the SLAM algorithms.
- The integration of the developed software into the codebase of the autonomous system team.
- The definition of interfaces with other autonomous system groups.
- The evaluation of the developed solution based on the defined metrics and test cases.

3.2.2 Constraints

The project also comes with several constraints that limit the scope and approach to the implementation of the SLAM algorithms. These constraints include:

- The software needs to be compatible with the Jetson Xavier AGX utilizing an ARM CPU architecture, the onboard computer of the autonomous vehicle.
- The software must run on Ubuntu 18.04, as this is the latest operating system supported by the Jetson Xavier AGX.
- Due to the limitation of Ubuntu 18.04, the latest version of ROS2 that can be used is Foxy Fitzroy.
- There have been issues with the accuracy and reliability of the LiDAR and GPS data in previous work done by the SLAM group, which may affect the performance of the SLAM algorithms.
- The Zed 2i camera is the only sensor available for collecting real-world data for testing and validation.
- Given that this is the first year all autonomous system groups are integrating their work, software integration will be a significant part of the project.
- As there is currently no one working on the perception group, additional work may need to be done to fill in the gaps.
- The project timeline is a limiting factor as well, with the thesis needing to be completed within 17 weeks.

In light of the defined requirements and constraints, a carefully planned and executed project management strategy is essential. Prioritizing tasks, allocating resources efficiently and maintaining regular communication between different groups will help mitigate risks and ensure the successful completion of the project.

3.3 Software

To implement multiple SLAM algorithms and also build, run, evaluate and test the system, a suitable software framework has to be developed. The following section will describe the different software artefacts which got identified as required in order to fulfil the mentioned requirements.

3.3.1 Simultaneous Localization and Mapping

SLAM algorithms fundamentally operate on sensor and control data that correspond to the same point in time. However, achieving such precision in synchronization is inherently challenging due to the variable data production rates across different types of sensors. To tackle this issue, a Signal Sync step is implemented. This phase estimates the measurement of a sensor at a given point in time by utilizing all available measurements from that sensor. This synchronized data is then utilized by the SLAM algorithm, which continuously generates a map containing the locations of all observed cones along with the current position of the race car.

Figure 3.2 provides an overview of the tasks that the system must fulfil, illustrating the role of Signal Sync and the SLAM algorithm within the larger problem context.

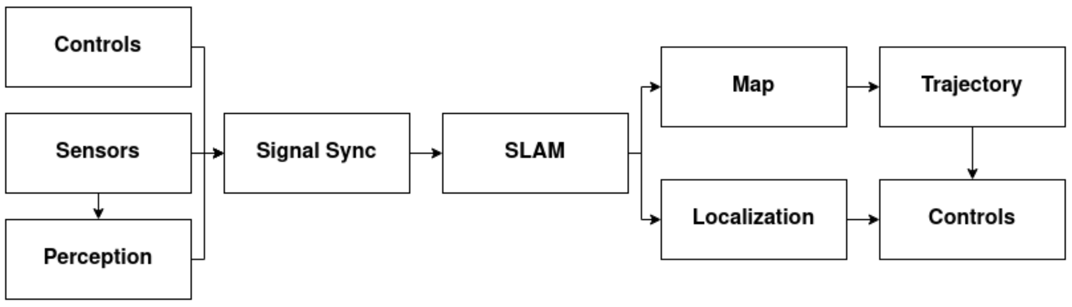


Figure 3.2: Division of the tasks that the system must fulfil

3.3.2 Software Pipeline

To achieve good results in the DC, it is important that all driverless sub-teams of ZUR work closely together. Integration of different software systems can be challenging and time-consuming. Besides the main goal of implementing a SLAM algorithm, the authors want to take responsibility and implement a solution to ease the integration process. To accomplish this, a software pipeline will be set up to automatically collect the code of all sub-teams when new code is added by a team and build it. Afterwards, tests should be run to verify the compatibility of the new code with the whole code base. As a last step, the built and tested software should be automatically deployed to the NVIDIA Jetson device. A conceptual approach for these requirements is presented in Figure 3.3. With this approach, a Minimal Viable Product (MVP) for the whole driverless software can be tested and easily iterated on. Therefore, the integration process can take place earlier in the project.

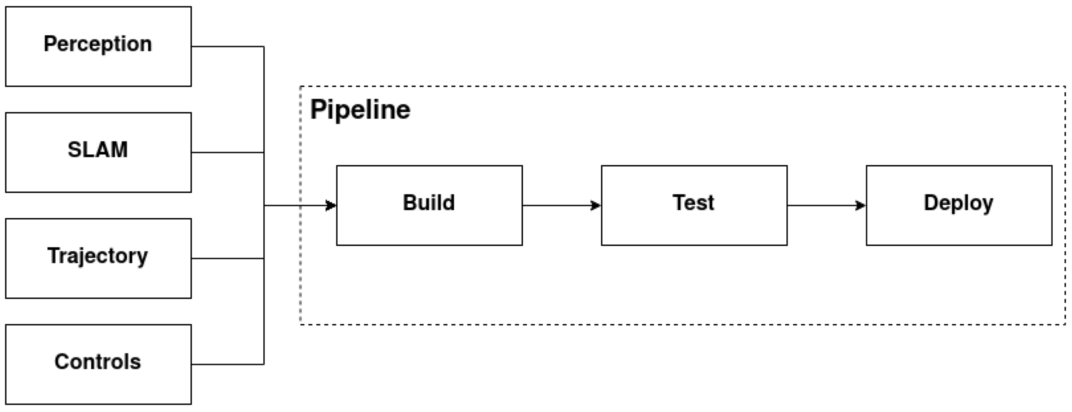


Figure 3.3: Concept of a software pipeline to automatically create, test and deploy new code of the team to the target hardware

3.4 Test Cases

Due to the fact that the new ZUR race car will only be finished towards the end of the project, the usage of mock data is required. In order to provide a sophisticated approximation of real-world data of the car, Gaussian noise should be added to all measurements provided to the SLAM algorithms. Different test cases were defined in Table 3.1, representing courses including the disciplines which will be conducted during the races in summer. Track Acceleration and Skidpad are replications of the original tracks at FS DC. Autocross is a self-designed track incorporating characteristics of the autocross event at FS DC. The courses Wide and Simple 0 were designed to provide concise and easily testable tracks.

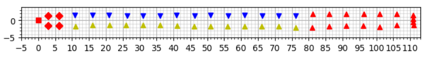
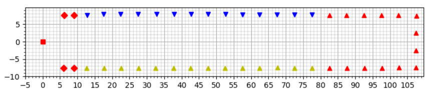
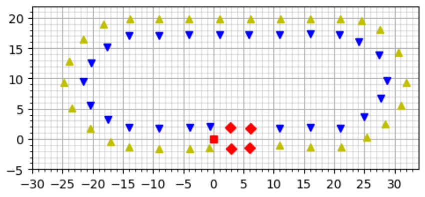
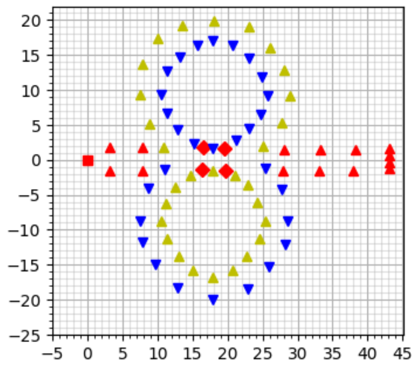
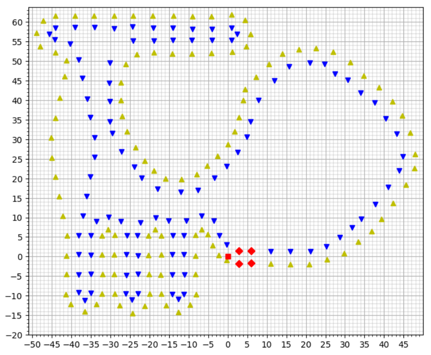
Name	Description	Track
Acceleration	The acceleration track is a straight line with a length of 75 m from starting line to finish line. The track is at least 3 m wide. Cones are placed along the track at intervals of about 5 m.	
Wide	The layout is the same as on the acceleration track with an increased track width of 15 meters.	
Simple 0	Track with two curves and two straight lines. The course is 3 meters wide and the curves have an inner diameter of 15.25 meters	
Skidpad	The skidpad track consists of two pairs of concentric circles in a figure-of-eight pattern. The inner circles are 15.25 meters in diameter. 16 cones are placed around the inside and 13 cones are positioned around the outside of each outer circle. The driving path is 3 meters wide.	
Autocross	The autocross track layout is a handling track including a big turn with 48 meters inner diameter, straights, hairpin turns with a minimum of 9 meters outside diameter and a track width ranging from 3 meters to 15 meters.	

Table 3.1: Test cases

3.5 Evaluation

It is essential to evaluate the performance of the various SLAM implementations adequately. The primary objective of the evaluation is to design and execute several representative test scenarios to assess the performance of different SLAM algorithms on different courses. This evaluation will involve gathering, quantifying and comparing data from these test runs by employing a set of comprehensive metrics designed to capture all potential sources of error.

Three major areas where discrepancies could emerge during SLAM operations were revealed. Firstly, the estimated position of the vehicle itself might deviate from its actual position due to noise or inaccuracies in the sensor data or algorithmic errors. This potential error is intrinsically connected to the second area of concern: the estimated positions of the cones. As these positions are inferred based on the vehicle's perceived position and sensor readings, they are also prone to inaccuracies due to sensor noise and computational errors. Lastly, the SLAM algorithms might either falsely identify non-existent cones or fail to detect real ones, both of which need to be considered in the evaluation.

In light of these potential error sources, three metrics were defined to evaluate the performance of the SLAM implementations:

- Mean Squared Error (MSE) of car position
- MSE of cone positions
- Number of cones (not) detected
- Number of cones wrongly detected

The MSE of the car position gets calculated using the mean squared error between the real and estimated car position of each time step.

To determine whether a cone got detected or not, a threshold is set to 1.5 meters indicating the maximum distance the estimation can have to the correct position to be considered correctly detected. Figure 3.4 shows possible use cases of the defined metrics. The figure shows the estimated cone positions (slightly greyed-out cones) as well as the correct cone positions. The green area around each real cone reflects the 1.5-meter threshold. Case A shows cones whose position got estimated too far away from the correct position which increases the number of not detected cones. Case B shows the standard case where a cone got estimated near the correct position. The red line shows the error which incorporates the MSE. Case C shows multiple cones detected in the 1.5 meters threshold. The nearest detected cone incorporates the MSE, all other cones in the radius of the defined threshold will be added to the number of wrongly detected cones.

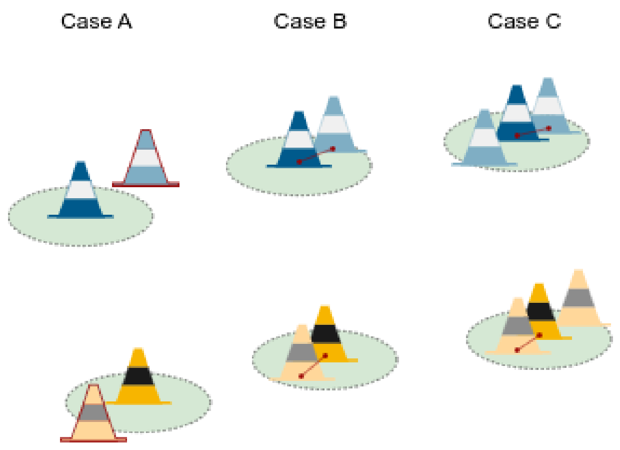


Figure 3.4: Visualizations of different possible cone detections by SLAM algorithms

The use of the MSE, a respected and widely-used measurement of prediction error, is particularly suitable for this evaluation. The MSE is sensitive to the magnitude of errors, penalizing larger deviations more heavily than smaller ones. This characteristic aligns well with our evaluation objectives, as more significant discrepancies in the estimated positions, whether of the vehicle or cones, could have a negative impact on the efficacy of trajectory planning.

3.6 Risk Analysis

Analyzing risks is a crucial step in any project as it provides foresight and mitigation strategies for potential problems that may arise. The risks can be broadly classified into software integration, mechanical issues, sensor malfunctions and data discrepancies.

3.6.1 Software Integration Risk

One of the primary risks pertains to the integration of the SLAM software with the other autonomous system groups. In a situation where this integration fails before the conclusion of this bachelor thesis, it may be challenging to rectify the issue before the competition in August. However, it is important to note that even in such a scenario, the defined research questions can still be answered. It is recommended to adopt a proactive approach by scheduling regular integration checks and setting early integration milestones to mitigate this risk.

3.6.2 Mechanical Issues

Mechanical issues with the car represent another significant risk. In the event of such a failure, it would not be possible to perform tests with the physical car. Fortunately, this risk would not significantly hamper the project as the main evaluation is based on simulated data. To avoid last-minute surprises, the autonomous system should be completely tested on the real car before the competition.

3.6.3 Sensor Malfunctions

Sensor malfunction is another risk factor that needs consideration. The inability of the sensors to function correctly would hinder testing with the actual car. However, similar to mechanical issues, the primary testing and evaluation for this project will be based on simulated data, reducing the impact of this risk. Regular sensor functionality tests should be performed to ensure they are working correctly and to identify and rectify any issues at an early stage.

3.6.4 Data Discrepancy

A significant risk that can affect the project's success in the competition is the discrepancy between simulated and real-world data. If the simulated data used for testing and evaluation does not accurately represent the real-world scenario, it could pose difficulties in parameterizing in the competition. Worse still, it could render the SLAM implementation entirely ineffective. Unfortunately, this risk might only be discovered after the thesis is completed when testing with the real car. To mitigate this risk, the simulated data should be as representative as possible of real-world driving conditions and, whenever possible, cross-verification with actual sensor data should be performed.

3.6.5 Other Risks

Additional risks might include unexpected changes in team composition, such as key members leaving or becoming unavailable. This can disrupt the progress and potentially delay the project. To manage this risk, tasks and responsibilities should be well-documented and distributed evenly among team members. Potential software bugs or algorithmic issues can also pose a risk. This risk can be mitigated through debugging, testing and code reviews.

In conclusion, while several risks could potentially affect the success of this project, proactive planning, regular testing and thorough review processes can help mitigate risks. The reliance on simulated data for testing and evaluation also provides a degree of flexibility and resilience against issues such as mechanical failures or sensor malfunctions. It is important to keep a frequent exchange with the other autonomous system groups to ease the integration for the competition.

Chapter 4

Implementation

4.1 Overview

In order to implement an effective autonomous driving system, an environment encompassing both hardware and software components is necessary. This chapter aims to delineate the core components and processes of our setup, starting from the selection of hardware elements to the establishment of a pipeline for streamlined software development.

The hardware components deployed in our autonomous car include a Jetson Xavier AGX and a ZED2i camera. We leverage the Robot Operating System (ROS) for creating our software structure. The heart of our project is the implemented SLAM algorithms. Lastly, we introduce our simulation environment, which plays an essential role in the testing and validation of our SLAM algorithms, supporting the progressive enhancement of our system.

4.2 Setup

In order to fulfil all the identified tasks, a proper development environment is needed. The key components assisting to achieve the goals in this thesis will be listed in this section.

4.2.1 Hardware

There are different hardware components plugged in the car which will be briefly described below.

Jetson Xavier AGX

NVIDIA provides a series of devices optimized for AI and Machine Learning (ML) applications called NVIDIA Jetson. They run their own Tegra System on Chip (SoC) platform using ARMv8.2 architecture. ZUR owns an NVIDIA Jetson Xavier AGX (as can be seen in 4.1), having 32GB of memory, an NVIDIA Carmel CPU and an NVIDIA Volta GPU containing 512 CUDA cores as well as 64 tensor cores. The Xavier AGX is one of the most powerful devices in the Jetson family.

Figure 4.1: Jetson Xavier AGX

ZED2i

The stereo camera owned by ZUR is a ZED2i, manufactured by Stereloabs. The ZED family is renowned in the robotics world and is used in many robotic applications. The camera comes with many sensors such as a gyroscope, barometer, magnetometer and temperature sensor as well as software features available using their SDK like object detection, SLAM and positional tracking.

The camera is mounted on the rollover bar behind and above the driver's head as illustrated in Figure 4.2a. It is connected to the Jetson located in the driverless box which is placed behind the driver seat. The setup can be seen in Figure 4.2.

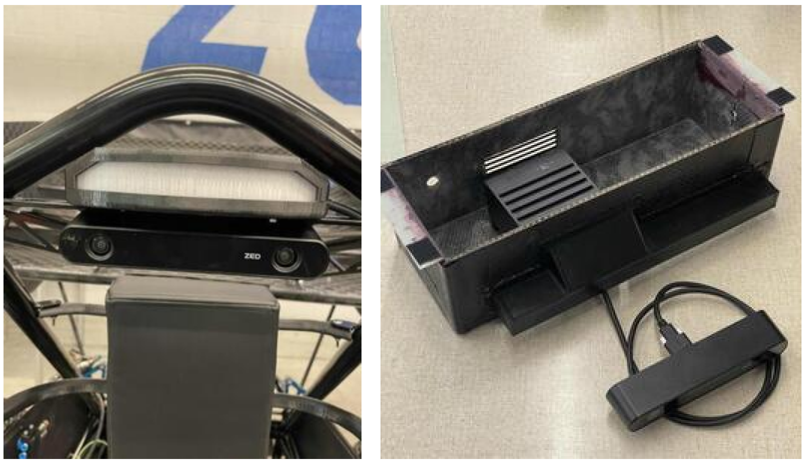

(a) The ZED mounted on the race car (b) The ZED together with the Jetson in the driverless box

Figure 4.2: The ZED2i stereo camera

4.2.2 Docker

Deploying finished code from the different teams on the Jetson can be done in multiple ways. Docker offers the possibility to create and test an image and as soon as its functionality gets verified, it can be deployed to multiple machines ensuring to have the same environment and the same libraries on every device. This functionality eliminates most environmental problems which can possibly happen when many individuals with different development environments want to run their code on different devices.

The Jetson was also equipped with the NVIDIA Container Runtime, enabling it to use all CUDA and Tensor features the Jetson's GPU offers from within containers.

4.2.3 Pipeline

As described in Section 3.3.2, a software pipeline should be set up in order to build, test and deploy all new code. Many cloud providers offer CI/CD functionality like software pipelines as well as standalone products to be self-hosted. Since for such a small pipeline costs on cloud infrastructure are low and maintenance of other students after this project will be easier, the pipeline was built on AWS with their own product CodePipeline. Figure 4.3 shows the different stages of the pipeline.

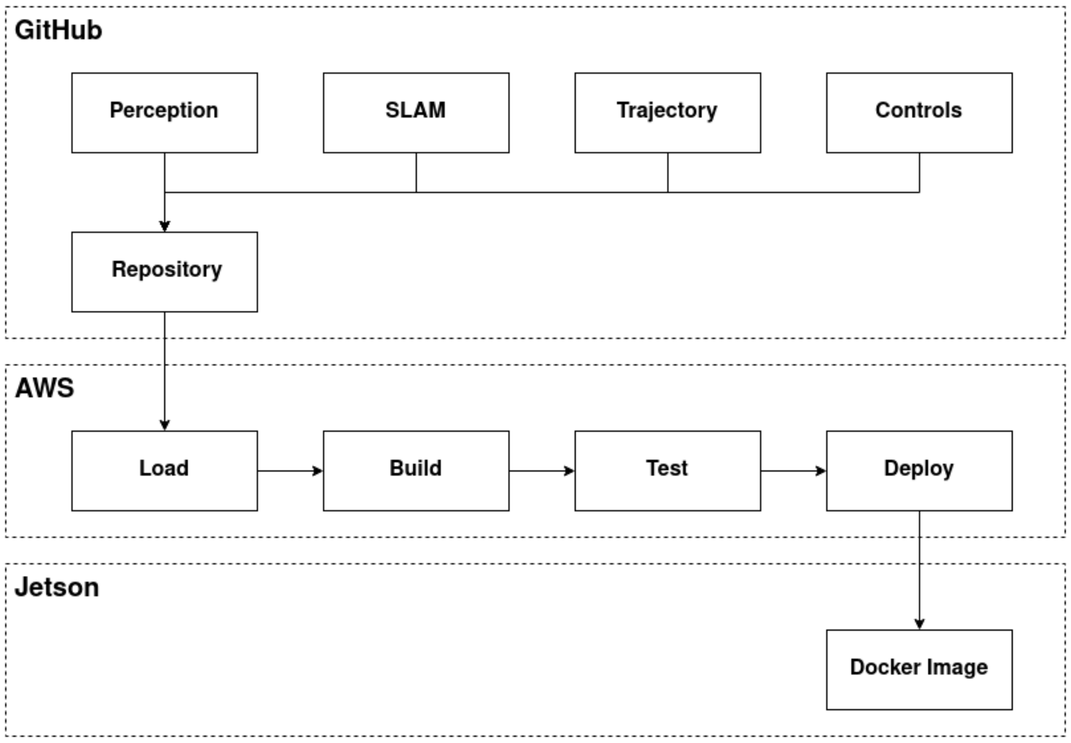


Figure 4.3: Steps of the established AWS pipeline

The codebase of ZUR already lies on GitHub so the decision was taken to leave it there and use GitHub in the course of the project. As soon as a branch from one of the driverless teams gets merged into the develop branch of the ZUR repository, the pipeline gets triggered.

In the first stage of the pipeline, the code from the GitHub repository gets cloned and provided as an artefact for the next stage.

The Build stage runs an existing docker base image containing all libraries and software components in order to run the code. In the container, the whole code gets built using the ROS2 build system colcon. In the Build stage, the pipeline builds the code on top of a custom-created docker base image containing all necessary software and drivers using a Dockerfile. Afterwards, the newly created docker image gets pushed to the AWS Elastic Cloud Registry (ECR).

The test stage uses the before-created docker image to run the whole system and runs checks whether everything works as expected. If all tests succeed, the image can be used on the Jetson. Unfortunately, the Jetson device is not always connected to the internet so as soon as new code is published and the tests were successful, the Jetson has to be connected to the internet manually in order to pull the latest Docker image.

4.2.4 Cloud Development Machine

For most people, undergraduate school is a time when financial resources are sparse. It is therefore very understandable that individuals who want to acquire skills in the field of computer vision cannot afford a computer running a CUDA-capable graphics card. Therefore, the team set up a development machine on AWS using their G5g-instances running on ARM processors which also have access to NVIDIA T4 graphics cards. This allows the team to have reasonable resources to test the algorithms when needed with an on-demand cost model saving costs compared to buying for example an NVIDIA-based eGPU.

4.2.5 ROS2

As described in Section 3.2, ROS2 has to be used in the course of this project. ROS2 is an open-source framework for robotic systems. It uses publishers and subscribers to enable communication between different components called nodes. Messages can be customized and published to so-called topics which can afterwards be subscribed to by other nodes. The various subteams of the ZUR driverless team all contribute to the software stack by adding a ROS2 node.

Stereolabs provides a ROS2 node for their ZED2i camera which publishes all configured data like the camera images, the depth map or the position of the camera.

4.3 Software Design

To be able to test different algorithms and change subparts we utilized abstraction in the software implementation. The general structure is illustrated in Figure 4.4.

A central controller initializes the ROS2 node and loads the configuration file. During initialization, the controller creates ROS2 subscribers for all topics needed like perception or the position of the car gathered from the ZED camera.

All gathered sensor data gets fed into the Sensor Model and with each perception message (meaning that the perception node has processed one camera image and detected all cones on it), the sensor data gets aligned to the timestamp of the perception message using the Synced Sensor Model.

After syncing the sensor data, the controller calls the update function of the configured SLAM algorithm which then performs its prediction and correction step as described in Section 2.3. For the prediction step, the outsourced Motion Model is used which predicts the position of the car based on the last known position and all available sensor data.

After the SLAM update step, the controller updates the Map Model holding the latest state of the map and also publishing it after an update, which can then be used by the trajectory node.

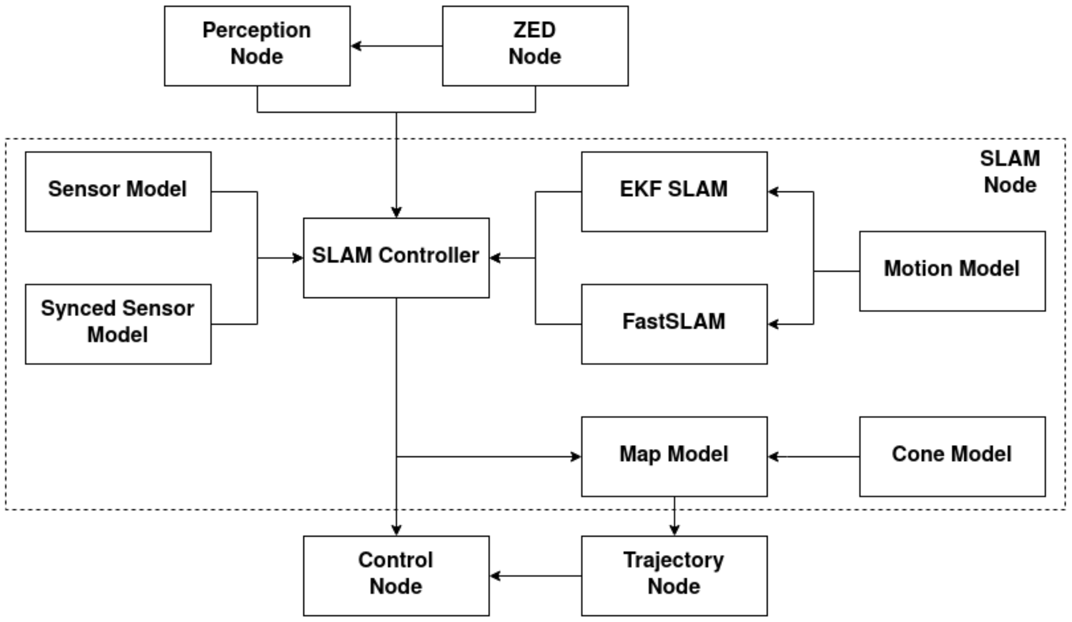


Figure 4.4: Architecture of the codebase

4.4 Simultaneous Localization and Mapping

The key software artefact of this project is the implemented SLAM algorithms. They will be described in more detail in the following sections.

4.4.1 Evaluation

The determination to utilize two specific SLAM algorithms, EKF SLAM and FastSLAM, resulted from an evaluation of current SLAM methodologies, as outlined in section 2.5.

The choice of EKF SLAM was driven by several compelling reasons. Firstly, its prior successful integration into multiple autonomous vehicles underscores its practical efficacy. Additionally, EKF SLAM's popularity in the field of SLAM solutions has led to a vast array of resources, textbooks and online material that expedite the implementation process. According to the literature, it is a robust and efficient algorithm, performing well under different environments and noise levels, further justifying its selection.

FastSLAM was the other chosen algorithm, inspired by its successful applications in various autonomous vehicles. Like EKF SLAM, FastSLAM is a prevalent SLAM algorithm, facilitating its implementation due to the abundance of supporting documentation. Furthermore, FastSLAM is recognized for its efficiency and scalability in handling larger environments, making it an attractive choice.

Despite GraphSLAM's initial consideration, the requirement for significantly higher computational resources made it less viable. Existing research indicated that GraphSLAM's computational demand is substantially greater than that of FastSLAM [31]. Based on these considerations, it was determined that our current hardware setup might struggle to process GraphSLAM effectively, leading to the decision to exclude it from our implementation plans.

4.4.2 Sensor Input

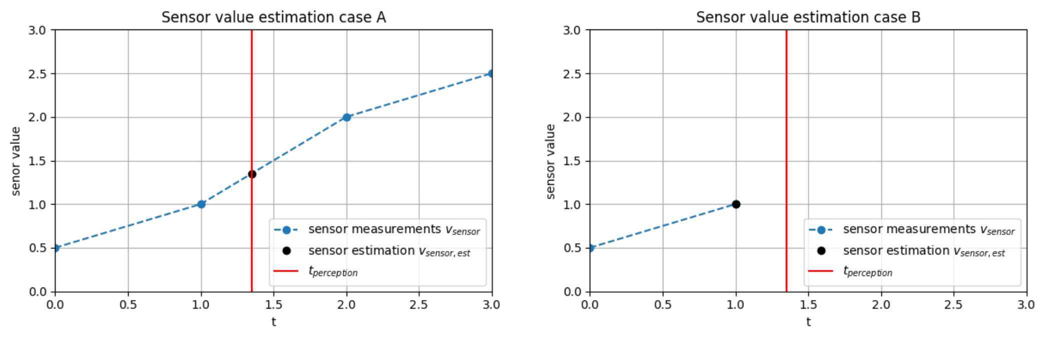
As described in Section 2.3, SLAM algorithms use input from controls like steering or pedals to predict the next position. Unfortunately, the ZUR race car from 2022 did not have any data of the controls which could have been consumed by the SLAM algorithms during development. On the other hand, the ZED camera provides an established positional tracking feature that returns the pose of the camera (position and orientation) relative to the starting point. Given these circumstances, the decision was taken to use the ZED pose estimation as the prediction of the current real pose.

Additionally, various sensors could be used in SLAM algorithms to improve the prediction of the current pose. Apart from the ZED stereo camera, ZUR is in possession of a GNSS as well as a LiDAR sensor. Based on feedback from team members and to prioritize a working MVP, the focus was set on the ZED stereo camera, leaving the GNSS and LiDAR sensor aside for future improvements of the system.

4.4.3 Sensor synchronization

Since sensor data gets published at different frequencies, sensor data needs to be synchronized. The SLAM algorithm in use runs every time a new observation of landmarks has been done, the trigger is a message sent by the perception node. Before running the SLAM algorithms, the controller generates a datapoint containing an estimation of all sensor values at the timestamp of the image used by the perception node. In Figure 4.4 the connection between the two sensor models as part of the SLAM node and the perception node can be seen.

The estimation of the sensor value at a given timestamp of the camera image $t_{perception}$ is done by creating a linear function between the two sensor measurements of v_{sensor} before and after timestamp $t_{perception}$ and calculation the value of the function at $t_{perception}$, which can be seen in Figure 4.5a. If no sensor measurement newer than $t_{perception}$ exists, the latest measurement of v_{sensor} is used, as can be seen in Figure 4.5b.



(a) Estimation when more recent sensor measurement is available

(b) Estimation when no more recent sensor measurement is available

Figure 4.5: Estimation of sensor value based on $t_{perception}$

This methodology is a very rudimentary abstraction of the problem but has shown so far to be a good trade-off between computational effort and accuracy. A possibility to improve if no newer measurement is available (as shown in Figure 4.5b) would be to use linear regression to create an estimation of the sensor value. Since this approach would be more resource-consuming and the sensor synchronization is performed after each perception update, the decision was made to keep the estimation as simple to calculate for the processor as possible.

4.4.4 Cone correspondence

After each prediction step, each SLAM algorithm tries to correct its map estimation based on the observed cones. During this process, a significant task is to determine whether an observed cone is one already seen before or one seen for the first time. Instead of calculating the Euclidian distance between two cones not considering the distribution of cones, the Mahalanobis distance (further illuminated in Section 2.3.4) was used.

The code calculates the Mahalanobis distance to all known cones. The observed cone is matched with the already known cone with the smallest distance to it, given that the distance is smaller than a configured threshold. If no distance is smaller than the threshold, the observed cone is considered a new cone.

4.4.5 EKF SLAM

The EKF SLAM algorithm is implemented according to the mathematical foundation depicted in section 2.2.5. The current implementation consists of the following functions:

- **Initialization:** Sets up the required models for the map, sensor data and motion as well as the logging functionality. Also initializes the state vector μ and the covariance matrix E.
- **Prediction:** Gets triggered by a perception message providing a list of detected cones z_t including their relative position and colour. Approximates the state $\overline{\mu}_t$ of the car based on all available data points from the mocked ZED positional data given the timestamp of the perception message.
- **Correction:** Gets the predicted state $\overline{\mu}_t$ and checks all received cones z_t if they are new or already seen using the Mahalanobis distance. Based on the result of this check and the predicted state $\overline{\mu}_t$, the state μ_t and the covariance matrix E get updated.
- **Publishing:** Broadcasts messages for the trajectory node and visualization tools using the map model.

The source code of the algorithm can be found in Appendix **??**.

4.4.6 FastSLAM

The FastSLAM algorithm uses a particle filter to represent the car's posterior belief about its pose and a set of landmarks. In our case, the landmarks are the cones of the tracks. Each particle in the filter is equipped with an individual map of cones. Our implementation consists of several methods representing different aspects of the FastSLAM algorithm:

- **Initialization:** Includes setting up the configuration parameters, sensor model, motion model, map model and a logger for debugging. The method also initializes a set of particles, with each particle representing a possible state of the robot, which includes the car's position and orientation (yaw) as well as the position of all cones.

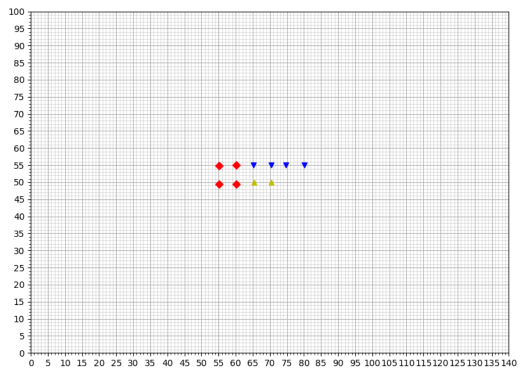
- **Prediction:** Each particle's state is predicted based on the car's motion model. The motion model simulates the movement of each particle according to the latest inputs and updates the particle's state accordingly.
- **Observation Update:** Uses the latest sensor observations to update the state of each particle. The method checks whether the observed cones have been seen before and, if so, updates their states. If a landmark has not been seen before, it is added as a new landmark. The weight of each particle is also updated based on how well the particle's predicted observations match the actual observations.
- **Resampling:** After updating all particles based on the observations, the resampling method is invoked to resample the particles. During resampling, a new set of particles is drawn from the current set. Each particle's probability of being chosen is proportional to its weight, which encourages diversity in the particle set and allows the algorithm to focus on the most likely states of the car.
- **Final State Estimation:** Estimates the final state of the car and the map by computing a weighted average of the states of all particles. The weights of the particles are normalized to ensure they sum up to one.

A number of helper methods are also included in the implementation to provide necessary computations for operations like angle normalization, weight normalization, coordinate transformations, landmark addition, landmark update, cone identification, weight computation, computation of Jacobians and the application of the Kalman Filter update.

The source code of the algorithm can be found in Appendix **??**.

4.4.7 Simulator

To thoroughly test and validate our SLAM algorithms, we utilized a simulation environment. This environment is composed of two main classes: DrawMap and DriveMap. The DrawMap class is responsible for generating different track configurations, providing control over the complexity and variability of the environments where the autonomous car operates. Conversely, the DriveMap class simulates the autonomous car's movement and sensor readings on a given track. Using our custom simulator, we established a comprehensive testing framework that allowed iterative refinement and validation of our SLAM implementation in a safe, cost-effective and time-efficient manner.


(a) In the process of creating a map

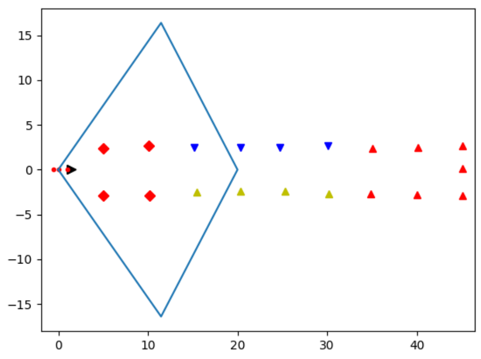

(b) Ready to race the track

Figure 4.6: Both simulator tools in action

Draw Map

The `DrawMap` class is used for constructing various racing tracks as seen in Figure 4.6a. It enables defining the placement of cones and the car's starting point interactively.

`DrawMap` initializes with lists corresponding to the four types of cones and the car's starting position. The `startDrawLoop` method launches the drawing loop, allowing the user to place cones and the car on the map by clicking. The user can change the mode (which object to place) by pressing 'q'. An `onclick` event, defined in the class, adds the clicked coordinates to the appropriate list based on the current mode. The `draw` method updates the map on the screen by plotting the points in different colours and shapes, depending on whether they represent a cone or a car and the type of cone they represent. The map is then saved into a YAML file.

Drive Map

The `DriveMap` class simulates the driving of a vehicle within a predefined map and incorporates the realistic physics of the car's motion. The car in this script is modelled using a simple bicycle model, a simplified representation of a car that effectively describes its motion. It represents a basic simulation in a controlled environment where the map and the car's perception of cones in the environment are defined in a YAML file. In Figure 4.6b the tool is illustrated.

The class reads a configuration file that includes parameters such as the perception angle of the car, the radius of its wheels, the frame rate, the car's wheelbase and the distance of the camera from the car's rear wheel. The car's operation is primarily controlled by the `writeDataLoop()` method. In this method, a keyboard listener is set up to accept specific key commands to control the car's acceleration and steering angle. The car is continuously simulated in a while loop, which updates the car's state, plots the car's state and the cones it perceives on a map and writes the state of various aspects of the car, including its GNSS position, Inertial Measurement Unit (IMU) data, perceived cone data, steering angle and Wheel Speed Sensor (WSS) data, into various text files. The `perceiveCones()` method calculates the relative positions of cones that the car can perceive based on its current position and orientation. This method checks whether a cone lies within the car's field of view.

4.4.8 Evaluator

The `Evaluator` script is designed to compare and visualize the results of a SLAM algorithm by comparing the real position of cones on a map against the calculated positions. The `coneRadius` attribute is set, defining the search radius for matching cones. The class logs various statistics about the cone data, including the total number of cones, the mean error and the number of offside cones. It also calculates the number of correct and offside cones respectively. The final plot with the real and calculated cones is saved as a PNG file in the specified output directory.

4.4.9 Interfaces

In order to communicate with the other driverless groups, communication interfaces are necessary. ROS2 offers the possibility to define custom messages which can later be published and consumed by nodes. Since the trajectory group requires specific update messages for cones in order to adjust the optimal path calculation, the message had to be designed accordingly. Therefore, a message was defined containing an id, the x and y coordinates, the type of the cone (blue cone, yellow cone, small red cone, large red cone) as well as an update flag indicating if a cone is new or already existing but its position got corrected. To reduce resource consumption, a threshold can be defined allowing only cones whose position change is greater than the threshold to be published using an update message.

Chapter 5

Results

5.1 Source Code

In the following sections, we will present the obtained results. All used code, as well as the raw data output from the different test cases, can be found on the ZHAW GitHub repository[1]. Since the repository is private, the file tree of the relevant parts of the system can be found in Appendix **??** as well as the implementation of EKFSLAM (Appendix **??**) and FastSLAM (Appendix **??**).

5.2 Test Cases

Both implementations were run on the defined test cases with the same path and the same collected information. The following section shows the results of these test runs.

The threshold for cones to be considered detected was set to 0.5 meters and is displayed as a circle around the real cone position in the figures below. The real cone positions are displayed with a dot and the estimates of the SLAM algorithms with a cross. The standard deviation of the Gaussian noise for the cone detections was set to 10 cm for the range and 0.1 rad for the bearing.

5.2.1 Acceleration

EKF

Figure 5.1 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.1 shows the metrics gathered during this test run.

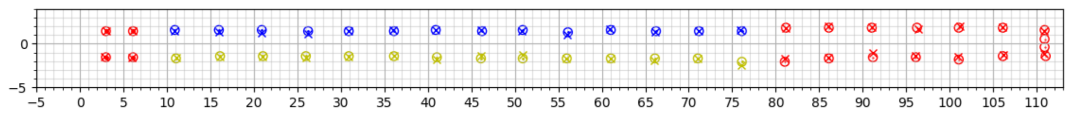


Figure 5.1: Evaluation of test case "Acceleration" using EKFSLAM

[1]https://github.zhaw.ch/FSZHAW/AutonomousSystem_ROS

Metric	Type	Calculated	Correct	MSE
Cones detected	Left cones (blue)	14	14	
	Right cones (yellow)	14	14	
	Large red cones	4	4	
	Small red cones	14	16	
Cone in radius	Left cones (blue)	14	14	0.0311
	Right cones (yellow)	14	14	0.0476
	Large red cones	4	4	0.0188
	Small red cones	14	16	0.0552
	Mean MSE (weighted)			**0.0424**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	0	14	
	Right cones (yellow)	0	14	
	Large red cones	0	4	
	Small red cones	0	14	

Table 5.1: Summary of test case metrics

It can be seen, that the test run did not produce any significant outliers. On the other hand, in the last column of small red cones located close to each other, only two of the four existing cones got detected which causes corresponding entries in Table 5.1 in the row "Cones detected".

FastSLAM

Figure 5.2 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.2 shows the metrics gathered during this test run.

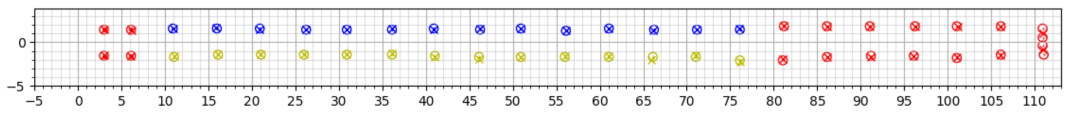


Figure 5.2: Evaluation of test case "Acceleration" using EKFSLAM

Metric	Type	Calculated	Correct	MSE
Cones detected	Left cones (blue)	14	14	
	Right cones (yellow)	14	14	
	Large red cones	4	4	
	Small red cones	14	16	
Cone in radius	Left cones (blue)	14	14	0.0056
	Right cones (yellow)	14	14	0.0233
	Large red cones	4	4	0.0099
	Small red cones	14	16	0.0729
	Mean MSE (weighted)			**0.0318**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	0	14	
	Right cones (yellow)	0	14	
	Large red cones	0	4	
	Small red cones	0	14	

Table 5.2: Summary of test case metrics

It can be seen, that the test run did not produce any significant outliers. On the other hand, in the last column of small red cones located close to each other, only two of the four existing cones got detected which causes corresponding entries in Table 5.2 in the row "Cones detected".

5.2.2 Wide

EKF

Figure 5.3 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.3 shows the metrics gathered during this test run.

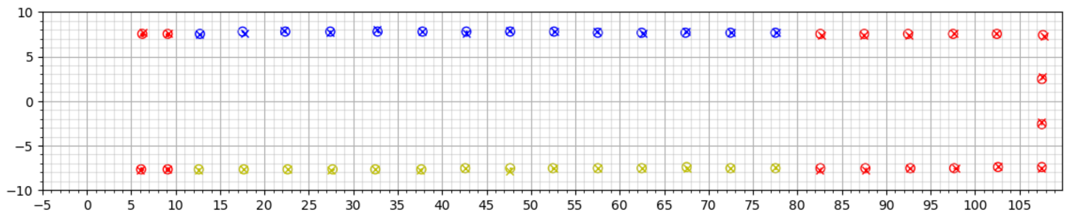


Figure 5.3: Evaluation of test case "Wide" using EKFSLAM

Metric	Type	Calculated	Correct	MSE
Cones detected	Left cones (blue)	14	14	
	Right cones (yellow)	14	14	
	Large red cones	4	4	
	Small red cones	14	14	
Cone in radius	Left cones (blue)	14	14	0.0243
	Right cones (yellow)	14	14	0.0186
	Large red cones	4	4	0.0201
	Small red cones	14	14	0.0294
	Mean MSE (weighted)			**0.0237**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	0	14	
	Right cones (yellow)	0	14	
	Large red cones	0	4	
	Small red cones	0	14	

Table 5.3: Summary of test case metrics

It can be seen, that the test run did not produce any significant outliers and all cones got detected within the defined threshold.

FastSLAM

Figure 5.4 shows the output graph of the evaluator displaying the estimated and real cone positions as well as the radius in which a cone would get considered detected. Table 5.4 shows the metrics gathered during this test run.

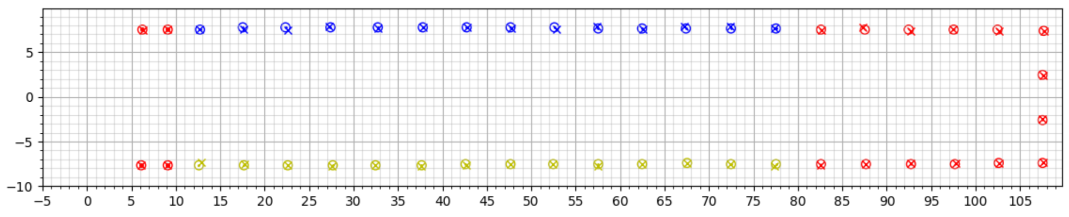


Figure 5.4: Evaluation of test case "Wide" using FastSLAM

Metric	Type	Calculated	Correct	MSE
Cones detected	Left cones (blue)	14	14	
	Right cones (yellow)	14	14	
	Large red cones	4	4	
	Small red cones	14	14	
Cone in radius	Left cones (blue)	14	14	0.0478
	Right cones (yellow)	14	14	0.0266
	Large red cones	4	4	0.0039
	Small red cones	14	14	0.0239
	Mean MSE (weighted)			**0.0302**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	0	14	
	Right cones (yellow)	0	14	
	Large red cones	0	4	
	Small red cones	0	14	

Table 5.4: Summary of test case metrics

It can be seen, that the test run did not produce any significant outliers and all cones got detected within the defined threshold.

5.2.3 Simple 0

EKF

Figure 5.5 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.5 shows the metrics gathered during this test run.

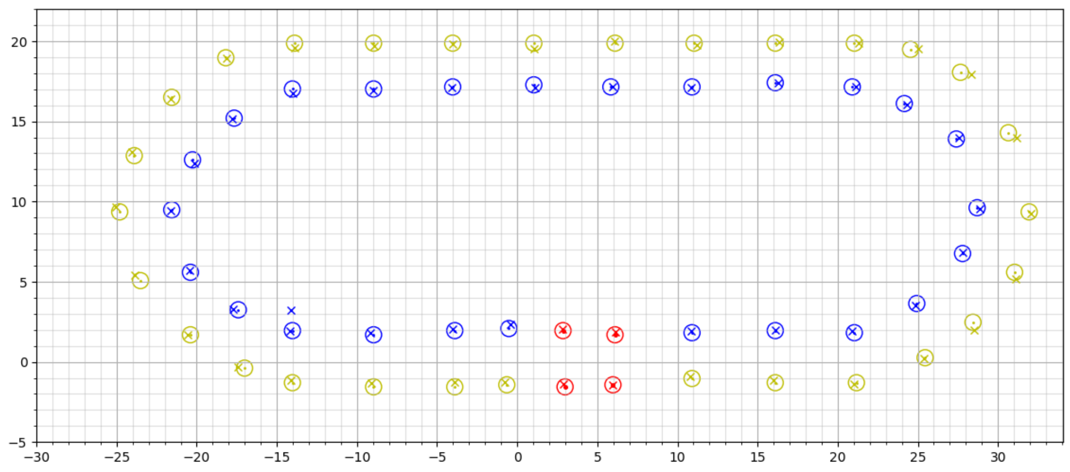


Figure 5.5: Evaluation of test case "Simple 0" using EKFSLAM

Metric	**Type**	**Calculated**	**Correct**	**MSE**
Cones detected	Left cones (blue)	26	25	
	Right cones (yellow)	29	29	
	Large red cones	4	4	
	Small red cones	0	0	
Cone in radius	Left cones (blue)	25	25	0.0333
	Right cones (yellow)	27	29	0.1032
	Large red cones	4	4	0.0217
	Small red cones	0	0	0.0000
	Mean MSE (weighted)			**0.0675**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	1	26	
	Right cones (yellow)	2	29	
	Large red cones	0	4	
	Small red cones	0	0	

Table 5.5: Summary of test case metrics

It can be seen, that the test run did not produce any significant outliers, apart from three cones (one blue, two yellow ones) in the lower-left and upper-right corner of the track. The cones are slightly outside of the defined threshold which causes corresponding entries in Table 5.5 in the row "Off side cones".

FastSLAM

Figure 5.6 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.6 shows the metrics gathered during this test run.

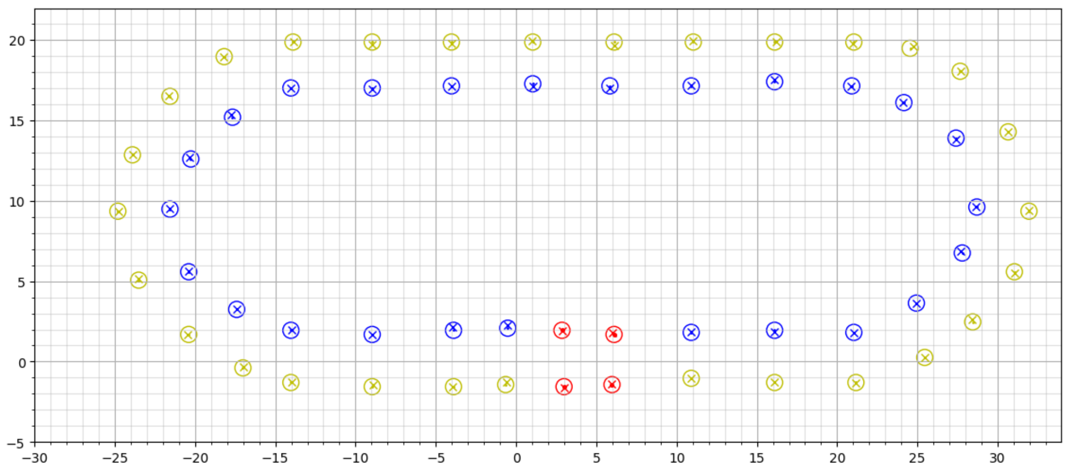


Figure 5.6: Evaluation of test case "Simple 0" using FastSLAM

Metric	Type	Calculated	Correct	MSE
Cones detected	Left cones (blue)	25	25	
	Right cones (yellow)	29	29	
	Large red cones	4	4	
	Small red cones	0	0	
Cone in radius	Left cones (blue)	25	25	0.0101
	Right cones (yellow)	29	29	0.0096
	Large red cones	4	4	0.0055
	Small red cones	0	0	0.0000
	Mean MSE (weighted)			**0.0096**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	0	25	
	Right cones (yellow)	0	29	
	Large red cones	0	4	
	Small red cones	0	0	

Table 5.6: Summary of test case metrics

It can be seen, that the test run did not produce any significant outliers and all cones got detected within the defined threshold.

5.2.4 Skidpad

EKF

Figure 5.7 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.7 shows the metrics gathered during this test run.

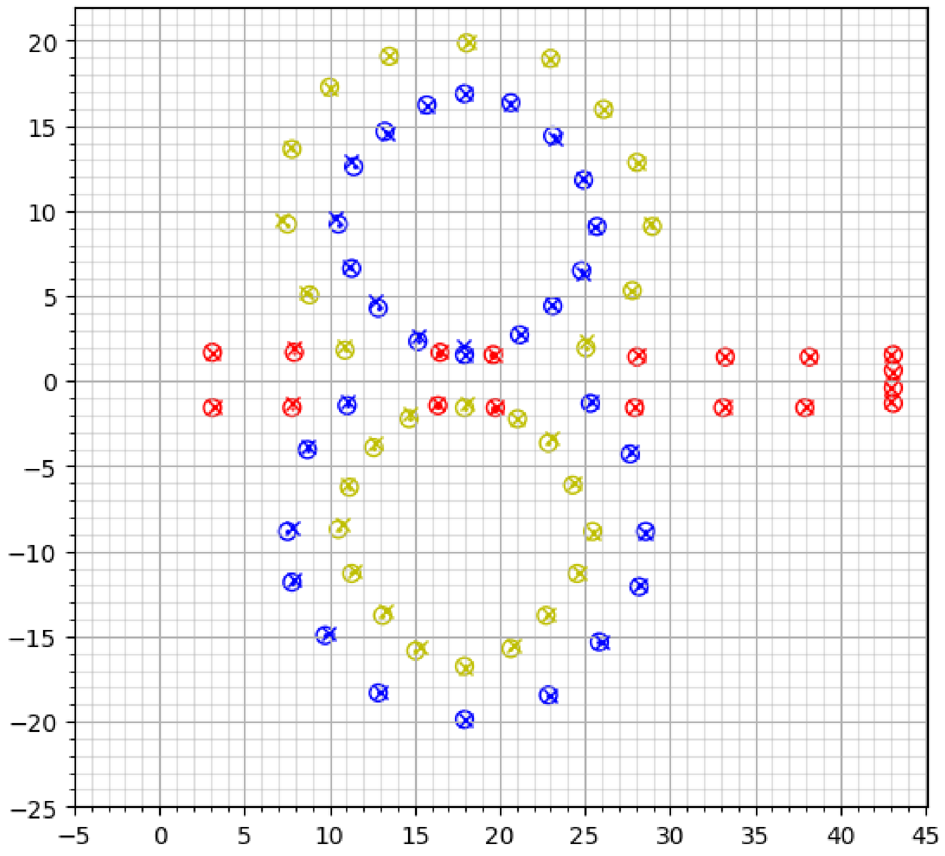


Figure 5.7: Evaluation of test case "Skidpad" using EKFSLAM

Metric	Type	Calculated	Correct	MSE
Cones detected	Left cones (blue)	30	29	
	Right cones (yellow)	29	29	
	Large red cones	4	4	
	Small red cones	14	14	
Cone in radius	Left cones (blue)	29	29	0.0511
	Right cones (yellow)	29	29	0.0536
	Large red cones	4	4	0.0113
	Small red cones	14	14	0.0162
	Mean MSE (weighted)			**0.0435**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	1	30	
	Right cones (yellow)	0	29	
	Large red cones	0	4	
	Small red cones	0	14	

Table 5.7: Summary of test case metrics

It can be seen, that the test run did not produce any significant outliers. One blue cone got detected twice in the center of the track between the two large red cones, which causes corresponding entries in Table 5.7 in the row "Cones detected" and "Off side cones".

FastSLAM

Figure 5.8 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.8 shows the metrics gathered during this test run.

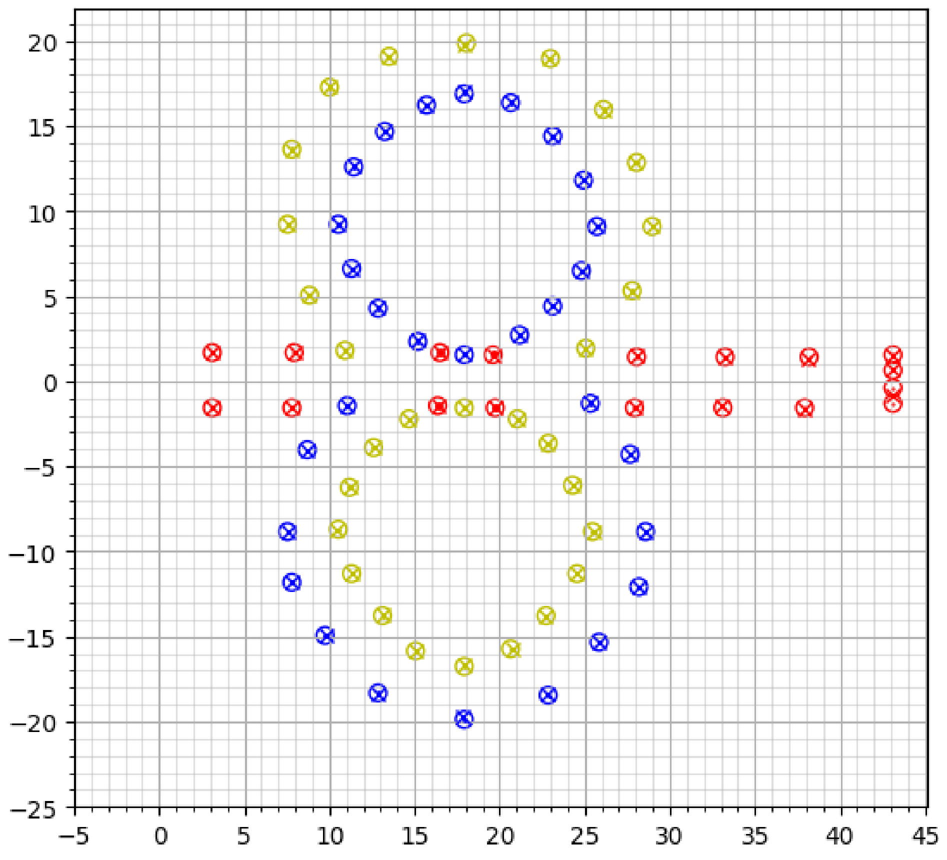


Figure 5.8: Evaluation of test case "Skidpad" using FastSLAM

Metric	**Type**	**Calculated**	**Correct**	**MSE**
Cones detected	Left cones (blue)	29	29	
	Right cones (yellow)	29	29	
	Large red cones	4	4	
	Small red cones	13	14	
Cone in radius	Left cones (blue)	29	29	0.0033
	Right cones (yellow)	29	29	0.0026
	Large red cones	4	4	0.0004
	Small red cones	13	14	0.0331
	Mean MSE (weighted)			**0.0084**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	0	29	
	Right cones (yellow)	0	29	
	Large red cones	0	4	
	Small red cones	0	13	

Table 5.8: Summary of test case metrics

It can be seen, that the test run did not produce any significant outliers. On the other hand, in the last column of small red cones located close to each other, only three of the four existing cones got detected which causes corresponding entries in Table 5.8 in the row "Cones detected".

5.2.5 Autocross

EKF

Figure 5.9 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.9 shows the metrics gathered during this test run.

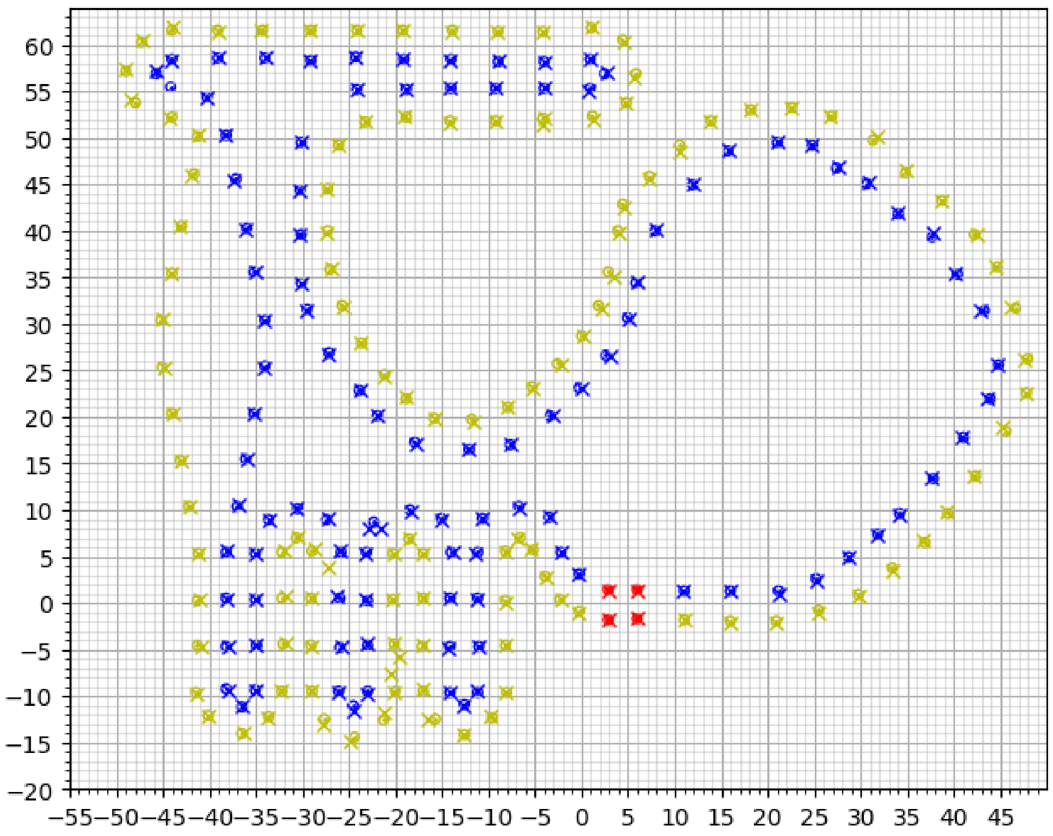


Figure 5.9: Evaluation of test case "Autocross" using EKFSLAM

Metric	**Type**	**Calculated**	**Correct**	**MSE**
Cones detected	Left cones (blue)	105	105	
	Right cones (yellow)	118	114	
	Large red cones	4	4	
	Small red cones	0	0	
Cone in radius	Left cones (blue)	103	105	0.0353
	Right cones (yellow)	111	114	0.0648
	Large red cones	4	4	0.0027
	Small red cones	0	0	0.0000
	Mean MSE (weighted)			**0.0497**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	3	105	
	Right cones (yellow)	14	118	
	Large red cones	0	4	
	Small red cones	0	0	

Table 5.9: Summary of test case metrics

It can be seen, that the test run did produce a reasonable amount of outliers, from which three are blue and 14 yellow. The blue outliers are located in the curvy section in the lower-left of the track and one in the middle region. The yellow outliers are spread over the course, five in the first section (large curve), two in the upper-left area and the remaining seven in the curvy section in the lower-left of the track.

Multiple cones got detected twice in the curvy section in the lower-left of the track, which causes corresponding entries in Table 5.9 in the row "Cones detected" and "Off side cones".

FastSLAM

Figure 5.10 shows the output graph of the evaluator displaying the estimated (cross) and real (dot) cone positions as well as the radius in which a cone would get considered detected. Table 5.10 shows the metrics gathered during this test run.

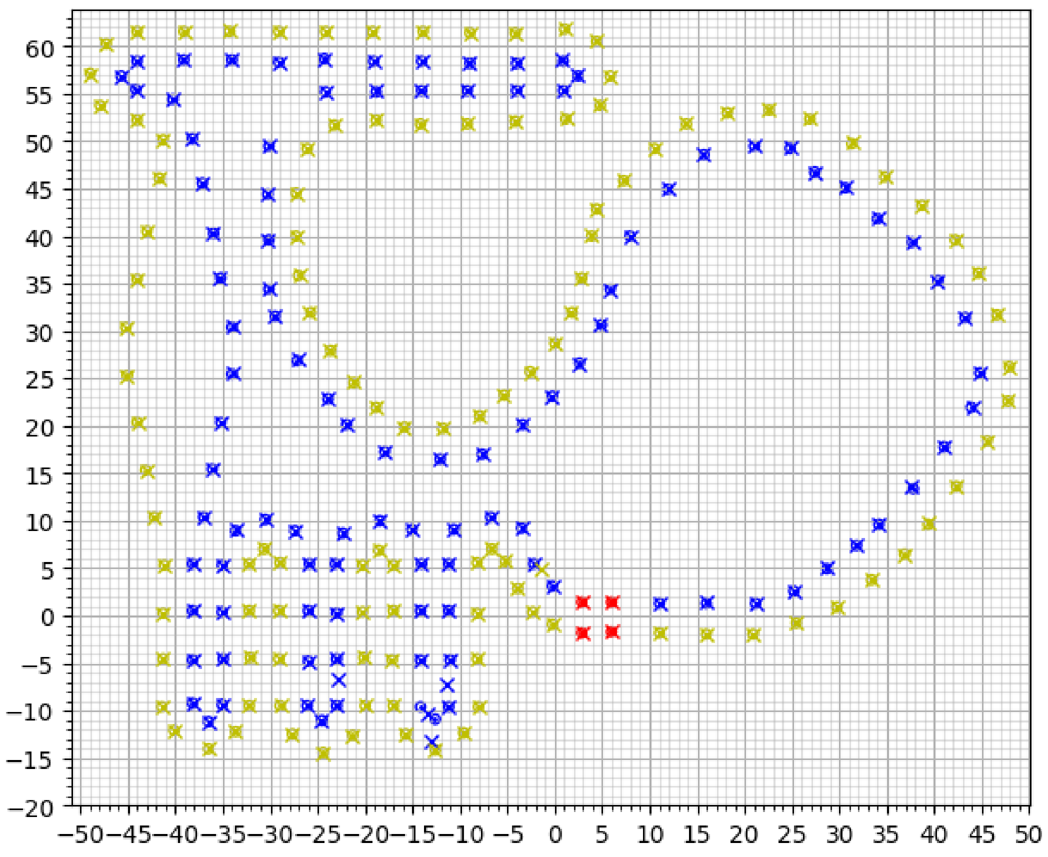


Figure 5.10: Evaluation of test case "Autocross" using FastSLAM

Metric	**Type**	**Calculated**	**Correct**	**MSE**
Cones detected	Left cones (blue)	107	105	
	Right cones (yellow)	115	114	
	Large red cones	4	4	
	Small red cones	0	0	
Cone in radius	Left cones (blue)	103	105	0.0056
	Right cones (yellow)	114	114	0.0030
	Large red cones	4	4	0.0011
	Small red cones	0	0	0.0000
	Mean MSE (weighted)			**0.0042**
		Off Side	**Calculated**	
Off side cones	Left cones (blue)	4	107	
	Right cones (yellow)	1	115	
	Large red cones	0	4	
	Small red cones	0	0	

Table 5.10: Summary of test case metrics

It can be seen, that the test run did produce a small number of outliers, from which four are blue and one is yellow. All outliers are located in the curvy section in the lower left of the track. The outliers cause corresponding entries in Table 5.10 in the row "Cones detected" and "Off side cones".

5.2.6 Summary

Table 5.11 shows a summary of all metrics collected during the test cases.

Test Case	**Algorithm**	**Detected**	**In radius**	**Off Side**	**Mean MSE**
Acceleration	EKFSLAM	36 / 38	36 / 38	0	0.0424
	FastSLAM	36 / 38	36 / 38	0	0.0318
Wide	EKFSLAM	36 / 36	36 / 36	0	0.0237
	FastSLAM	36 / 36	36 / 36	0	0.0302
Simple 0	EKFSLAM	59 / 58	56 / 58	3	0.0675
	FastSLAM	58 / 58	58 / 38	0	0.0096
Skidpad	EKFSLAM	77 / 76	76 / 76	1	0.0435
	FastSLAM	75 / 76	75 / 76	0	0.0084
Autocross	EKFSLAM	227 / 223	218 / 223	17	0.0497
	FastSLAM	226 / 223	221 / 223	5	0.0042

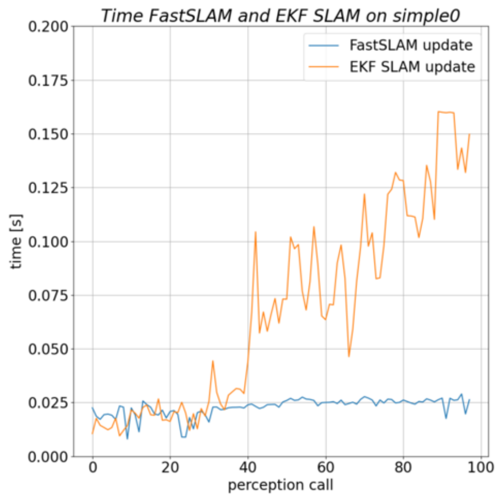
Table 5.11: Summary of metrics of all test cases

Both algorithms showed similar results in the straight tracks "Acceleration" and "Wide". Both algorithms seem to struggle with cones standing close to each other, as can be seen in the test cases having a line of small red cones at the end of the track.

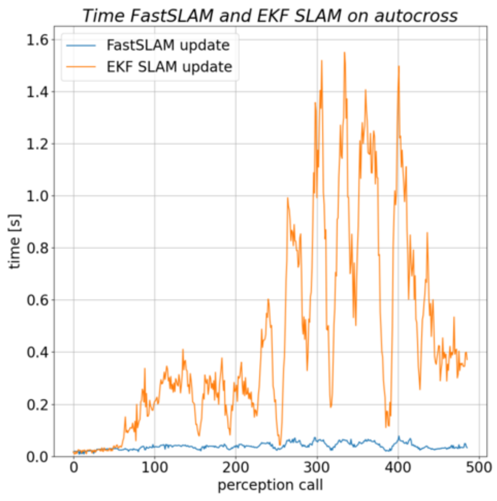
The Mean MSE metric shows, that FastSLAM detects cones with a significantly higher accuracy while having fewer Off Side detections on longer tracks consisting of a large number of cones, namely "Simple 0", "Skidpad" and "Autocross".

5.3 Time Efficiency Analysis

The computational duration required for executing the update step of each algorithm was measured on two distinct tracks: simple0 and autocross.



(a) Computation time for update steps on simple0



(b) Computation time for update steps on autocross

Figure 5.11: Execution duration comparison between EKF SLAM and FastSLAM during update step

In Figure 5.11, the computational duration for each update step is depicted graphically. In both cases, the occurrence of each perception triggers an algorithm update step. For the simple0 track in Figure 5.11a, a total of 98 perceptions were processed by both algorithms, while for the autocross track seen in Figure 5.11b, the algorithms processed 486 perceptions. At perception 319 four cones were sent by the perception node and EKF SLAM processed the information in 0.19 seconds while in perception 335 the perception node send 25 cones and EKF SLAM needed 1.55 seconds. FastSLAM needed 0.03 seconds in perception 319 and 0.06 in perception 335.

Chapter 6

Discussion

6.1 Overview

In this chapter, an opinion about the reached results is provided. A lot has been achieved in this 17-week project, but there are various imperfections we would like to address. Because of the limited time available, we focused on the key components to achieve a solid MVP to compete in this year's FS DC.

6.2 SLAM algorithms

6.2.1 Control and Sensor Input

As described in section 2.3, SLAM algorithms primarily use input from the car controls to estimate the position of the car which can be improved by additionally using measurements from other sensors like IMU, GNSS or WSS. As described in section 4.4.2, the current implementation only uses the ZED positional tracking feature to estimate the current position. Using the controls of the car and fusing multiple available sensor measurements may improve the estimation and therefore the whole SLAM implementation.

6.2.2 EKF SLAM

EKF SLAM has shown to be a simple and easy-to-implement SLAM algorithm. On short tracks the performance was sufficient. Whereas on Autocross the performance was inadequate. Due to computational inefficiencies as illustrated in Section 5.3, the algorithm did not manage to calculate the course in a time that would be applicable in a real race. As described in the background in Section 2.2.5, this is due to its mathematical principle updating the covariance matrix of all landmarks against each other during each update run and clearly showing EKF SLAMs limits.

6.2.3 FastSLAM

FastSLAM performed on all tracks satisfactorily and in an efficiency that is viable in a race setting as computational effort stays constant over time. Illustrated in Section 5.3 the time needed for each update step does not rise with larger tracks. The whole autonomous system with its various demanding software systems needs a high amount of computational resources. Therefore a time-efficient implementation of SLAM is desirable. Regarding accuracy the calculated MSE were on three of five tracks lower than the introduced sensor noise. This represents the capabilities of particle filters in this use case.

6.2.4 Narrow Cones

Both algorithms struggled to discern cones standing close to each. For example in the acceleration or Skidpad track at the end of the course, the four cones are one meter apart. As seen in Section 5.2 these cones are fused together and detected as one cone. However, setting the SLAM parameters to values allowing the detection of all narrow cones can cause situations where one cone gets detected twice because of sensor noise. This would cause problems in other situations. Detecting two of the four cones at the end of the track is not critical, given that there is no specific need to detect all these cones since the car has to stop in these sections.

6.2.5 Uncertainty Parameters

The results presented results on the test courses with Gaussian error. The SLAM parameters for uncertainty were adjusted accordingly to fit this type of data best. However, we are aware that using real sensor data from the race car, the parameters certainly have to be tuned to fit the uncertainty of the used sensors best which will cause additional effort before we can compete in FS DC.

6.3 Sensor synchronization

The current implementation of the sensor synchronization, as described in section 4.4.3, estimates the sensor value of a given point in time, especially for cases where only older measurements are available, legitimating it by saving computational resources. The system may be improved by slightly optimizing the estimation function for this case while continuously observing the impact on resource consumption. Particularly when using rarely updating sensors such as GNSS, usually updating once per second, the current estimation function will most probably affect the system negatively in a manner which enforces an improvement of the estimation function.

6.4 Simulator

An efficient and straightforward simulator was implemented using a 2D bicycle model, which has been valuable for generating data. However, despite the success and utility of the current simulator, the potential benefits of utilizing a more advanced tool were recognized. The IAMP has developed a sophisticated simulator, offering an appealing alternative due to its 3D capabilities and realistic physics. The integration of such features could provide a more comprehensive, realistic, and nuanced testing environment for our algorithms, thereby contributing to the optimization of the whole autonomous driving system.

6.5 Metrics

To discuss the second research question of the thesis, defined in section 1.4, the defined metrics have to be evaluated. This is conducted visually by determining whether the resulting metric values of a test case also match the visual quality of the result.

Reporting the number of cones estimated within a defined radius around the correct position has proved to be helpful in quickly seeing the general performance of a test run.

Using the MSE as a metric to quantify the quality of estimated cone positions performs satisfactorily. However, during the test iterations, it became clear that cones estimated too far from the course will negatively impact trajectory planning more than estimations too far away from the course. This is due to the fact that cones in the course narrow down the available space for the car to pass possibly leading to enforced evasion maneuvers, whereas cones away from the course do not require adjustments in the trajectory. Therefore, a metric considering this fact by punishing cone error in the direction of the course more than error away from the course is worthwhile.

The last metric numbering the off-side cones compared to all detected cones are generally feasible. However, testing showed that cones not detected affect trajectory planning less than cones detected twice for similar reasons as mentioned above. The trajectory can be left unchanged missing one cone, whereas additional cones can affect trajectory planning. This leads to the conclusion, that the metric is also to be improved to meet this restriction.

While on all test cases including Autocross, the resulting metrics were sufficient regardless of the time needed for the update steps of each Algorithm. It is crucial for a race to have live information on the cones ahead while driving. This is not represented in the current metrics. An additional metric that represents the live aspect of the algorithms is therefore required.

Chapter 7

Conclusion and Outlook

During this project, a reliable framework able to automatically evaluate the SLAM algorithm used with self-designed test cases has been created. The high level of modularity allows easy integration of new SLAM algorithms as well as automated evaluation using test cases custom created by the developed simulator. Two promising SLAM algorithms were identified to be suitable and implemented for the given use case. Multiple metrics were defined and showed reliable results correlating with visual assessments. The constructed test cases showed, that the algorithms deliver solid results with incorrect detections only appearing rarely on long courses. FastSLAM showed reliable results with constant consumption of computational resources, whereas EKF SLAM required an increasing amount of processor time during the course, negatively impacting the performance on long courses. Given these insights, we decided to focus on improving FastSLAM in order to verify its performance on the real car with real sensor data to finally use it later this year during the FS DC. All in all, it can be said that a considerable amount of value was added to the driverless system of ZUR, providing together with the scientific results of this thesis a solid foundation for further projects in this highly interesting field.

We are deeply grateful for the opportunity to engage in such an interesting project and feel blissful for all the lessons learned.

After all, we identified plenty of possibilities to improve in the context of the implementation as well as the thesis itself during the course of this project. The most significant topics are set out in the following list:

- Conduct a questionnaire with other FS teams about their implementations
- Use controls and additional sensor data from the car
- Implement GraphSLAM to evaluate its performance against FastSLAM
- Integrate the simulation software developed by IAMP
- Keep an eye on end-to-end deep learning approaches
- Introduce additional metrics and evaluate the current solutions

Bibliography

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016.

[2] Chris Anderson. New dyson robot vacuum has 360-degree slam camera tech, Sep 2014.

[3] Marcus Andersson and Martin Baerveldt. Simultaneous localization and mapping for vehicles using orb-slam2. *Chalmers Univ. of Technology, Gothenburg, Sweden*, 2018.

[4] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *arXiv preprint arXiv:1805.12177*, 2018.

[5] Pinglei Bao, Liang She, Mason McGill, and Doris Y Tsao. A map of object space in primate inferotemporal cortex. *Nature*, 583(7814):103–108, 2020.

[6] Johannes Betz, Hongrui Zheng, Alexander Liniger, Ugo Rosolia, Phillip Karle, Madhur Behl, Venkat Krovi, and Rahul Mangharam. Autonomous vehicles on the edge: A survey on autonomous vehicle racing. *IEEE open journal of intelligent transportation systems*, 3:458–488, 2022.

[7] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[8] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017.

[9] Axel Brunnbauer and Markus Bader. „traffic cone based self-localization on a 1: 10 race car,“. *Traffic Cone Based Self-Localization on a 1: 10 Race Car*, 2019.

[10] Sergen Cansiz. Multivariate outlier detection in python, Mar 2021.

[11] Le Chang and Doris Y Tsao. The code for facial identity in the primate brain. *Cell*, 169(6):1013–1028, 2017.

[12] Jie Chen, Jian Sun, and Gang Wang. From unmanned systems to autonomous intelligent systems. *Engineering (Beijing, China)*, 12(5):16–19, 2022.

[13] Jun Cheng, Liyan Zhang, Qihong Chen, Xinrong Hu, and Jingcao Cai. A review of visual slam methods for autonomous driving vehicles. *Engineering Applications of Artificial Intelligence*, 114:104992, 2022.

[14] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon,

U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[15] Lars B Cremean, Tully B Foote, Jeremy H Gillula, George H Hines, Dmitriy Kogan, Kristopher L Kriechbaum, Jeffrey C Lamb, Jeremy Leibs, Laura Lindzey, Christopher E Rasmussen, et al. Alice: An information-rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics*, 23(9):777–810, 2006.

[16] Pranav Dixit. Mercedes beats tesla in self-driving, becomes first certified level-3 autonomous car company in us, Jan 2023.

[17] H.F. Durrant-Whyte. Uncertain geometry in robotics. *IEEE Journal on Robotics and Automation*, 4(1):23–31, 1988.

[18] Hugh Durrant-Whyte, David Rye, and Eduardo Nebot. Localization of autonomous guided vehicles. In Georges Giralt and Gerhard Hirzinger, editors, *Robotics Research*, pages 613–625, London, 1996. Springer London.

[19] Dale C. Ferguson, Joseph C. Kolecki, Mark W. Siebert, David M. Wilt, and Jacob R. Matijevic. Evidence for martian electrostatic charging and abrasive wheel wear from the wheel abrasion experiment on the pathfinder sojourner rover. *Journal of Geophysical Research*, 104(E4):8747–8759, 1999.

[20] Dimitrios Geromichalos, Martin Azkarate, Emmanouil Tsardoulias, Levin Gerdes, Loukas Petrou, and Carlos Perez Del Pulgar. Slam for autonomous planetary rovers with global localization. *Journal of Field Robotics*, 37(5):830–847, 2020.

[21] Riccardo Giubilato, Wolfgang Stürzl, Armin Wedler, and Rudolph Triebel. Challenges of slam in extremely unstructured environments: The dlr planetary stereo, solid-state lidar, inertial dataset. *IEEE Robotics and Automation Letters*, 7(4):8721–8728, 2022.

[22] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[23] Andrew J. Hawkins. Cruise continues to burn gm's cash as robotaxis expand to daylight hours, Apr 2023.

[24] Laurens Hobert, Andreas Festag, Ignacio Llatser, Luciano Altomare, Filippo Visintainer, and Andras Kovacs. Enhancements of v2x communication in support of cooperative autonomous driving. *IEEE communications magazine*, 53(12):64–70, 2015.

[25] Sae International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE international*, 4970(724):1–5, 2018.

[26] Peraphan Jittrapirom, Valeria Caiati, Anna Maria Feneri, Shima Ebrahimigharehbaghi, María J Alonso-González, and Jishnu Narayan. Mobility as a service: A critical review of definitions, assessments of schemes, and key challenges. *Urban Planning*, 2(2):13–25, 2017.

[27] Sheena A Josselyn and Susumu Tonegawa. Memory engrams: Recalling the past and imagining the future. *Science*, 367(6473):eaaw4325, 2020.

[28] Juraj Kabzan, Miguel I. Valls, Victor J. F. Reijgwart, Hubertus F. C. Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, Ankit Dhall, Eugenio Chisari, Napat Karnchanachari, Sonja Brits, Manuel Dangel, Inkyu Sa, Renaud Dubé, Abel Gawel, Mark Pfeiffer, Alexander Liniger, John Lygeros, and Roland Siegwart. Amz driverless: The full autonomous racing system. *Journal of field robotics*, 37(7):1267–1294, 2020.

[29] kyburz. Buddy mobility builds on kyburz technology, 2018.

[30] Nathan Lambert, Louis Castricato, Leandro von Werra, and Alex Havrilla. Illustrating reinforcement learning from human feedback (rlhf). *Hugging Face Blog*, 2022. https://huggingface.co/blog/rlhf.

[31] Nick Le Large, Frank Bieder, and Martin Lauer. Comparison of different slam approaches for a driverless race car. *tm - Technisches Messen*, 88(4):227–236, 2021.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[33] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4:333–349, 1997.

[34] Yi Ma, Doris Tsao, and Heung-Yeung Shum. On the principles of parsimony and self-consistency for the emergence of intelligence. *Frontiers of Information Technology & Electronic Engineering*, 23(9):1298–1323, 2022.

[35] Markus Maurer, Barbara Lenz, and J. Christian Gerdes. *Autonomous Driving : Technical, Legal and Social Aspects.* Springer Open, Cham, 2016.

[36] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[37] L Daniel Metz. Potential for passenger car energy recovery through the use of kinetic energy recovery systems (kers). Technical report, SAE Technical Paper, 2013.

[38] Borna Monazzah Moghaddam and Robin Chhabra. On the guidance, navigation and control of in-orbit space robotic missions: A survey and prospective vision. *Acta Astronautica*, 184:70–100, 2021.

[39] Umberto Montanaro, Shilp Dixit, Saber Fallah, Mehrdad Dianati, Alan Stevens, David Oxtoby, and Alexandros Mouzakitis. Towards connected autonomous driving: review of use-cases. *Vehicle system dynamics*, 57(6):779–814, 2019.

[40] Henry J. Moore, Donald B. Bickler, Joy A. Crisp, Howard J. Eisen, Jeffrey A. Gensler, Albert F. C. Haldemann, Jacob R. Matijevic, Lisa K. Reid, and Ferenc Pavlics. Soil-like deposits observed by sojourner, the pathfinder rover. *Journal of Geophysical Research*, 104(E4):8729–8746, 1999.

[41] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4316–4336, 2020.

[42] NASA. Touchdown! NASA's mars perseverance rover safely lands on red planet, February 2021.

[43] NASA. Mars 2020 perseverance rover - overview, n.d.

[44] NASA. Mars helicopter - ingenuity, n.d.

[45] Felix Nobis, Johannes Betz, Leonhard Hermansdorfer, and Markus Lienkamp. Autonomous racing: A comparison of slam algorithms for large scale outdoor environments. In *Proceedings of the 2019 3rd international conference on virtual and augmented reality simulations*, pages 82–89, 2019.

[46] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[47] Adrian Padeanu. Bmw confirms level 3 self-driving system to launch this year on 7 series, Jan 2023.

[48] Vardan Papyan, XY Han, and David L Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

[49] Francisca Rosique, Pedro J Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors (Basel, Switzerland)*, 19(3):648–, 2019.

[50] Emma Roth. Waymo can now provide fully driverless rides in san francisco, Nov 2022.

[51] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[52] G. Savage. Formula 1 composites engineering. *Engineering Failure Analysis*, 17(1):92–115, 2010. Papers presented at the 25th meeting of the Spanish Fracture Group.

[53] Asif A Siddiqi. *Beyond Earth: A chronicle of deep space exploration, 1958-2016*, volume 4041. National Aeronautis & Space Administration, 2018.

[54] Robert Sim, Matt Griffin, Alex Shyr, and James J Little. Scalable real-time vision-based slam for planetary rovers. In *IEEE IROS Workshop on Robot Vision for Space Applications*, pages 16–21, 2005.

[55] Randall Smith, Matthew Self, and Peter Cheeseman. *Estimating Uncertain Spatial Relationships in Robotics*, pages 167–193. Springer New York, New York, NY, 1990.

[56] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[57] Formula Student Spain. Formula student spain rules, 2023.

[58] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. *Advances in neural information processing systems*, 30, 2017.

[59] Statista. Robotics - worldwide, n.d.

[60] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

[61] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[62] Victor Talpaert, Ibrahim Sobh, B Ravi Kiran, Patrick Mannion, Senthil Yogamani, Ahmad El-Sallab, and Patrick Perez. Exploring applications of deep reinforcement learning for real-world autonomous driving systems. *arXiv preprint arXiv:1901.01536*, 2019.

[63] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics.* MIT Press, Cambridge, Mass., 2005.

[64] Jessica Van Brummelen, Marie O'Brien, Dominique Gruyer, and Homayoun Najjaran. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation Research Part C: Emerging Technologies*, 89:384–406, 2018.

[65] Kristian Wahlqvist. A comparison of motion priors for ekf-slam in autonomous race cars, 2019.

[66] Brian Wilcox, Larry Matthies, Donald Gennery, Brian Cooper, Tam Nguyen, Todd Litwin, Andrew Mishkin, and Henry Stone. *Robotic vehicles for planetary exploration*, volume 1. Springer, 1992.

[67] Yun-Hua Wu, Zhi-Cheng Yu, Chao-Yong Li, Meng-Jie He, Bing Hua, and Zhi-Ming Chen. Reinforcement learning in dual-arm trajectory planning for a free-floating space robot. *Aerospace Science and Technology*, 98:105657, 2020.

[68] Lei Yan, Wenfu Xu, Zhonghua Hu, and Bin Liang. Multi-objective configuration optimization for coordinated capture of dual-arm space robot. *Acta Astronautica*, 167:189–200, 2020.

[69] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *arXiv.org*, 2020.

Chapter 8

Lists

List of Figures

List of Tables

List of Listings

Acronyms

AI Artificial Intelligence. 5, 6, 8, 21, 23, 36

AMZ Academic Motorsports Club Zurich. 23, 25

CV Internal Combustion Engine Vehicles. 1

DC Driverless Cup. 1, 3, 26, 28, 30, 57, 58, 61

DV Driverless. 2, 3, 27

EKF Extended Kalman Filter. 16–20, 24–26, 57

EV Electric Vehicles. 1

FS Formula Student. 1–3, 24, 26–28, 30, 57, 58, 61, 71

GA Globally Adapted. 10

GNC Guidance, Navigation, and Control. 10

GNSS Global Navigation Satellite System. 12, 40, 44, 57, 58

HAD High Accuracy Digital. 9

IAMP Institute of Applied Mathematics and Physics. 2, 58, 61

IMU Inertial Measurement Unit. 44, 57

KIT Karlsruhe Institute of Technology. 24

LiDAR Light Detection and Ranging. 6, 7, 26, 29, 40

LLM Large Language Models. 22

MaaS Mobility-as-a-Service. 8

ML Machine Learning. 36

MSE Mean Squared Error. 32, 46–53, 55, 57, 58

MVP Minimal Viable Product. 30, 40, 57, 79

PDF Probability Density Function. 17

RBPF Rao-Blackwellized Particle Filter. 10

RL Reinforcement Learning. 10

RLHF Reinforcement Learning from Human Feedback. 22, 23

ROS Robot Operating System. 35

SAE Society of Automotive Engineers. 5, 6

SIFT Scale-Invariant Feature Transform. 10

SLAM Simultaneous Localization and Mapping. 2, 3, 5, 7, 9, 10, 18–20, 27–33, 35, 40, 43, 45, 57, 58

SoC System on Chip. 36

V2I Vehicle-to-Infrastructure. 7

V2V Vehicle-to-Vehicle. 7

WSS Wheel Speed Sensor. 44, 57

ZUR Zurich University of Applied Sciences Racing. 1–3, 26, 30, 36–38, 40, 61

Appendix A

Project Management

Table A.1 gives a rough overview of the defined milestones and their due dates.

Milestone	**Due date**	**Description**
MS 1	02.03.23	Software pipeline takes code from team repository and creates running image for target hardware
MS 2	13.04.23	Minimal viable product is able to process sensor data and produce a map that can be used by the trajectory team
MS 3	20.04.23	MVP can be automatically evaluated on defined test cases and defined metrics
MS 4	11.05.23	Initial solution is improved and alternatives are implemented
MS 5	18.05.23	All implementations are evaluated in defined test cases and defined metrics
MS 5	01.06.23	The report is finished

Table A.1: Defined milestones and due dates of the project

It was also agreed that over the course of the project, regular meetings would be held weekly to update the supervisors about the status and discuss problems and possible remedial actions.